目录

前言

MATLAB 是科学与工程中一个非常广泛使用的计算工具，不管你的背景是什么——物理、化学、数学还是工程学——它都适合你。学习一款数学计算工具有三个好处：第一，如果你是手工完成计算的，那么它就像一个后台检查工具。假如你是学生，有一个工具可以检查你的答案总是好的。我并不是说你要过份依赖某种计算工具，好像它就是神谕一样。如果你的教授要求你手工完成工作，之后你就可以使用 **MATLAB** 或其它计算工具来检验你的工作。

第二个原因是用 **MATLAB** 这样一个工具来绘图和进行数学运算是非常有价值的，你不用花费很多时间来手工绘图，**MATLAB** 就可以帮你产生出所需要的非常漂亮的图形。

第三个原因是在某种程度上你的职业要求你使用数学计算工具。如果你是一个做理论研究的教授，有时候你所做的工程用解析法行不通，如果你在工厂或实验室工作，可能会碰到有些工作无法用手工完成。**MATLAB** 已经在大学、实验室或公司中广泛应用，懂得 **MATLAB** 将在你的简历加上重要的一项。

一句话，本书是直接针对于 **MATLAB** 初学者，目的也不是教专家使用 **MATLAB** 去解决复杂问题，相反，本书是介绍给 **MATLAB** 新人，使他们进入数学计算世界。这里要介绍的是使用 **MATLAB** 去解决某些基本问题——绘函数的图形、解代数方程、计算积分和解微分方程，所以要本书的例子较简单，针对新手。如果你以前从没接触过 **MATLAB**，或者是在使用 **MATLAB** 有很多疑问，那么本书将帮助你学到一些基本技巧，使你能够掌握 **MATLAB**。本书仅是掌握 **MATLAB** 的垫脚石。

致谢

感谢 Rayjan Wilson 对本书给了全面细心的审阅，他深刻见解的评论和详细审阅对本书的成功出版至关重要。

第一章



MATLAB 环境

我们从着眼于 **MATLAB** 的用户界面开始我们的 **MATLAB** 之旅。在我们干劲十足地解决数学问题之前，先学习如何输入命令，创建文件及其它常用的必须知道的工作。本章所讨论的内容会在整本书中用到，也会贯穿你以后学习使用 **MATLAB** 的过程。在本书中会涉及到开始使用 **MATLAB** 的一些基础知识，我们的目的是在每一章告诉你一些基本知识，你可以使用这些知识解决一些重要问题。读完本书后，你还不会成为 **MATLAB** 专家，但能够自由地使用 **MATLAB** 并且完成不少常见任务，在学习上取得进步或在工作上为进一步学习打下基础。无论如何，我们都得来看看 **MATLAB** 启动后的主界面。

用户界面概述

本书我们假定你使用 Windows——虽然对本书的大部分内容并没有什么影响。请注意我们使用的是 **MATLAB** 7.1 版本。

MATLAB 的启动与其它 Windows 程序一样，点击开始－程序，找到 **MATLAB** 文件夹，你就会看到几项——取决于你的安装，但至少有如下几项

- **MATLAB** 7.1
- M-file Editor
- R14SP3Uninstaller

选择 **MATLAB**(7.1)启动程序，屏幕上显示的 **MATLAB** 默认界面如图 1.1 所示，可以看到，屏幕被划分成三大块，它们是

- 当前目录（Current Directory）
- 历史命令窗口（Command History）
- 命令窗口（Command Window）

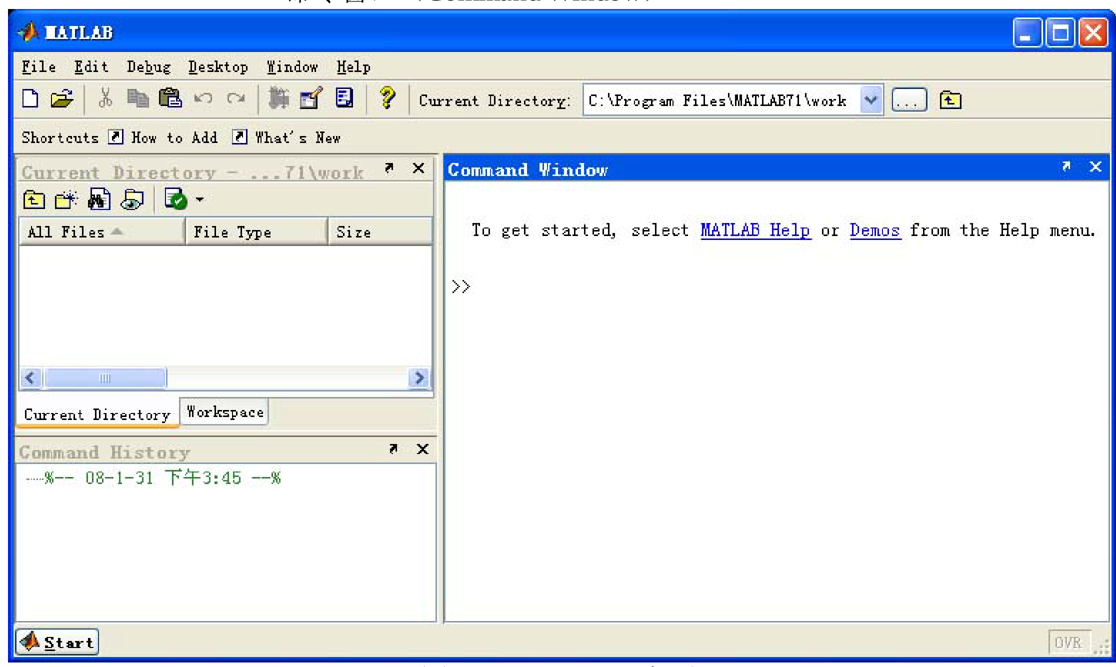


图 1.1 MATLAB 桌面

MATLAB 桌面顶部的标准菜单允许你做管理文件和调试文件等工作，你可能已经注意到右边有一个下拉列表框，它可以设置当前工作路径，不过这里最重要的是命令窗口。

命令窗口与算法基础

命令窗口位于 **MATLAB** 桌面的右边，命令在双大于号“>>”提示符后面输入

```
>>
```

这里我们开始输入一些基本命令。假如你想知道一些数字表述式的值，简单的输入就可以了。如我们想知道 433.12 乘以 15.7 的结果，在提示符后面输入 433.12 * 15.7 然后按 Enter 回车，结果如下：

```
>> 433.12*15.7
ans =
  6.8000e+003
```



译者注：

1.命令窗口中须手工输入的内容，本书使用蓝色字体，但注释内容使用绿色字体。

2.原本上式输入完成后的显示情况应为

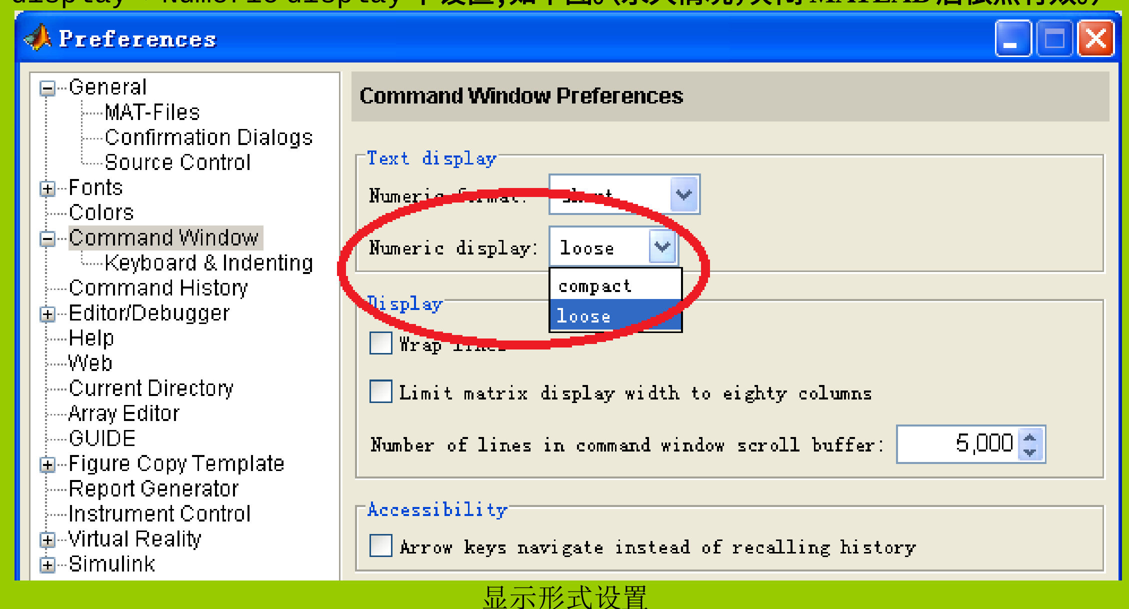
```
>> 433.12*15.7

ans =

  6.8000e+003

>>
```

为使显示紧凑，本书去掉其中的空行。如读者想使命令窗口内容也以紧凑形式显示，可先输入 `format compact` **（临时情况，关闭 MATLAB 后无效，要取消的话，输入** `format` **或** `format loose` **即可）或在菜单** `File->Preferences->Command Window->Test display->Numeric display` **中设置，如下图。（永久情况，关闭 MATLAB 后依然有效。）**



显示形式设置

3.如果想清除命令窗口的内容，可使用 `clc` **命令；如果想消除工作空间的所有变量，可输入** `clear` **命令。**

MATLAB 立即输出答案，并命名为 ans——这是一个**变量**或**符号**名，用来表示结果。如果我们想要使用自定义的变量名，例如变量 x，假设我们想要让它等于 5 乘以 6，则输入如下：

```
>> x=5*6
x =
    30
```

一旦变量输入系统，我们就可以在后在面使用它，假设我们要计算 x 乘以 3.56 的值，并把它赋给 y。输入

```
>> y = x * 3.56
y =
  106.8000
```

现在，你可能已经注意到在这个例子中，我们在方程每项之间都留有空格，这样提高了我们输出的可读性，看起来更专业些。**MATLAB** 并不要求这些空格，我们可以输成 `y=x*3.56` 或者 `y = x * 3.56`，不过后者更清楚明了。当你的表达式比较复杂时，带上空格就显得非常重要了。推荐这么做！

我们总结一下 **MATLAB** 输入基本算法。要写两数相乘 ab，在 **MATLAB** 中我们输入

```
a * b
```

两数相除 $\frac{a}{b}$，输入为

```
a / b
```

这种除法被称为**右除**，**MATLAB** 也允许另一种写法，叫**左除**。如果我们要计算 $\frac{b}{a}$，我们可以使用反斜杠代替斜杠，表示反过来除，表达式如下：

```
a \ b
```

幂a^b 以下面的形式输入

```
a ^ b
```

最后，相加和相减以普通形式输入即可

```
a + b
a - b
```

MATLAB 运算符的优先级与数学中优先级一致，不过要注意左除与右除的情况：幂运算优先于乘和除，右除优先于左除，加和减的优先级最低，如果想改变优先级，用圆括号括起来。

例 1-1

使用 **MATLAB** 计算

$$5\times\left(\frac{3}{4}\right)+\frac{9}{5} \quad 和 \quad 4^3\times\left[\frac{3}{4}+\frac{9}{2\times 3}\right]$$

解 1-1

第一个表达式的命令为

```
>> 5*(3/4) + 9/5
ans =
    5.5500
```

对于第二个表达式，我们在 $a \wedge b$ 形式的数值之后使用了括号。虽然是简单表达式，我们分部输入变量，得到

```
>> r = 4^3
r =
    64
>> s = 3/4 + 9/(2*3)
s =
    2.2500
>> t = r * s
t =
   144
```

赋值运算符

符号“＝”称为**赋值运算符**，不过，在 **MATLAB** 中有时候把它理解为**把值赋给一个变量的指令**更为恰当一些。这种理解的差异可以用下面方式显示出来。如果你输入

```
>> x + 6 = 90
```

在 **MATLAB**，你会得到下面的响应

```
??? x + 6 = 90
          |
Error: The expression to the left of the equals sign is not a valid
target for an assignment.
```

可见，在纸上写的完全正确的代数表达式 **MATLAB** 会完全不知道如何处理。不过，如果你把 90-6 **赋**给变量 *x*，**MATLAB** 会高兴完成，此时写法是

```
x = 90 – 6
```

赋值运算符在计算机程序中处理起来更像是赋值指令的另一个例子是递归赋值给变量，例如，如果变理已经定义，**MATLAB** 允许你写成

```
x = x + 4
```

下面的语句是完全正确的

```
>> x = 34^2
x =
       1156
>> x = x + 4
x =
       1160
```

在赋值运算符右边使用变量，必须事先给变量赋值，因此，下面的表达式会产生错误

```
>>  x = 2
x =
    2
>> t = x + a
??? Undefined function or variable 'a'.
```

下面的表达式则不会产生错误

```
>> x = 2
x =
    2
>> a = 3.5
a =
   3.5000
>>  t = x + a
t =
   5.5000
```

在很多时候，我们并不需要 **MATLAB** 输出结果，这时只需要在表达式后面加上分号(;)即可。在下面的命令中，开始我们输入 `x = 3`， **MATLAB** 及时地报告结果，第二次我们输入 `x = 3;` ，**MATLAB** 就没有再花费空间输出结果，而是直接跳到命令提示符，等待下次输入。

```
>> x = 3
x =
    3
>> x = 3;
>>
```

我们还可以在一行中包含多个表达式。例如，下面的表达式是合法的。

```
>>  x = 2; y = 4; z = x*y
z =
    8
```

注意那两个分号，它们告诉 **MATLAB** 我们不想看到 x 和 y 的值。

当做许多计算时，结果可能会产生大量变量，可以通过在命令窗口中输入 *who* 来刷新内存，告诉 **MATLAB** 显示到目前为止所有变量名称。例如，在我们的例子中，我们得到

```
>> who

Your variables are:

V    a    ans  r    s    t    x    y    z
```

如果输入 *whos*，我们会得到更多信息，告诉我们当前内存中的变量，类型，每个变量所分配的内存空间，以及它们是否是**复数**（见下面）。在我们的例子中我们有

```
>> whos
  Name      Size                    Bytes  Class

  V         1x1                         8  double array
  a         1x1                         8  double array
  ans       1x1                        16  double array (complex)
  r         1x1                         8  double array
  s         1x1                         8  double array
  t         1x1                         8  double array
  x         1x1                         8  double array
  y         1x1                         8  double array
  z         1x1                         8  double array

Grand total is 9 elements using 80 bytes
```

现在假设我们要全部重新开始，要这样做，我们输入 *clear* 命令。要清除全部变量只需输入 *clear* 然后回车即可，要清除特定变量，则在 clear 后面带上变量名列表。如果我们要清除使用过的变量 x、y 和 z，我们输入

```
>> clear x y z
>>
```

此时 **MATLAB** 返回命令提示符，没有给出任何提示。但如果你想再使用这些变量而没有再赋值，它们就好像没有存在过一样。

较长的表达式可以在行尾加上三点（...）省略号进行续行输入。例如

```
>> FirstClassHolders = 72;
>> Coach = 121;
>> Crew = 8;
>> TotalPeopleOnPlane = FirstClassHolders + Coach ...
+ Crew
TotalPeopleOnPlane =
   201
```

泽者注：虽然在输入过程中使用 Shift+Enter 也能换行输入，但这已经不是同一个命令。如下（注意 Coach 后面没有省略号）：

```
>> TotalPeopleOnPlane = FirstClassHolders + Coach
+ Crew
TotalPeopleOnPlane =
   193
ans =
     8
```



Coach 后面带省略号的那行定义了变量 *TotalPeopleOnPlane*，当你输入省略号后回车，**MATLAB** 会把光标移到下一行等待用户更进一步的输入。

好了，到这里你可能会像我开始 **MATLAB** 时一样，想知道如何控制 **MATLAB** 在屏幕上显示的数字。目前我们的例子中，**MATLAB** 输出小数点后四位，这在 **MATLAB** 中称为 *short* 格式，是 **MATLAB** 的默认格式。如果这个精度已经满足你的要求，那么就没有必要改变它。如果要更多位数，就必须用格式命令告诉 **MATLAB** 在小数点后输出更多位。如果要用 16 位代替 4 位，输入 *format long*。想知道如何工作，看看下面用两种格式显示的计算例子。

```
>> format long
>> x = 3 + 11/16 + 2^1.2
x =
   5.98489670999407
>> format short
>> x = 3 + 11/16 + 2^1.2
x =
    5.9849
```

比较 *long* 和 *short* 格式，注意 *short* 格式在第四位四舍五入进位到 9。如果要进行财务计算，你可以使用 *format bank* 格式命令。正如所预计，所有数字被取到小数点后两位。

```
>> format bank
>> hourly = 35.55
hourly =
         35.55
>> weekly = hourly * 40
weekly =
       1422.00
```

MATLAB 使用指数形式显示大数值，即是把 5.4387×10^3 表示成 5.4387e + 003。你也可以让所有的数字都以这种风格显示。这种风格也可以使用 short 或 long 格式来定义，对于 short（小数点后四位）输入 format short e。要使小数点后 15 位加上指数，输入 format long e，这里我们用短指数格式举例

```
>> format short e
>> 7.2*3.1
ans =
  2.2320e+001
```

如果我们输入 *format rat*，**MATLAB** 将自动查找最接近结果的比例式，是不是相当有意思？下面我们重复前一个计算

```
>> format rat
>> 7.2*3.1
ans =
     558/25
```

基本数学定义式

MATLAB 附带了许多基本的或者是常见的数学常量和函数，我们用例子来演示

例 1-2

计算半径为 2m 的球的体积

解 1-2

球体的体积由下式决定

$$V = \frac{4}{3}\pi R^3$$

MATLAB 当然预定义了 π，要使用的话只需输入 *pi*。在定义了表示半径的变量之后，我们就可以输入下面的表达式得到体积

```
>> r = 2;
>> V = (4/3) * pi * r^3
V =
   33.5103
```

另一个在很多数学应用中闻名的常数是 e=2.718。在 **MATLAB** 中我们可以引用 *e*，输入 *exp*(*a*)得到 e^a 的值。这里有些例子

```
>> exp(1)
ans =
    2.7183
>> exp(2)
ans =
    7.3891
```

要得到一个数字的平方根，我们输入 *sqrt*，例如

```
>> x = sqrt(9)
x =
     3
>> x = sqrt(11)
x =
    3.3166
```

要得到 *x* 的**自然对数**，输入 *log*(*x*)

```
>> log(3.2)
ans =
    1.1632
>> x = 3;log(5)
ans =
    1.6094
```

如果要得到以 10 为底的对数，输入 *log*10(*x*)

```
>> x = 3; log10(x)
ans =
    0.4771
```

MATLAB 还带有基本三角函数及反三角函数，默认以弧度为参数，以小写标准形式输



入即可，例如

```
>> cos(pi/4)
ans =
    0.7071
```

要使用反三角函数，在三角函数名前加 *a*。例如，要计算一个数的反三角，格式如下

```
>> format rat
>> atan(pi/3)
ans =
    1110/1373
```

复数

在 **MATLAB** 中我们也可以输入复数。提醒一下，-1 的平方根定义为

$$i = \sqrt{-1}$$

复数可以写成 $z=x+iy$ 的形式，其中 x 是 z 的实部，y 是 z 的虚部。在 **MATLAB** 中输入复数很容易，默认就把 i 当为负一的平方根。例如

```
a = 2 +3i
b = 1 - i
=> a + b = 3 + 2i
```

让我们在 **MATLAB** 中验证一下。在 **MATLAB** 中可以不用在 i 前面添加空格或乘号(*)。

```
>> format short
>> a = 2 + 3i;
>> b = 1 - i;
>> c = a + b
c =
   3.0000 + 2.0000i
```

修正输入

有时候我们输入表达式时会带有错误，当你按 ENTER 回车后才意识到，这时没必要重新输入整行，只需使用方向键向上移动，修正错误，然后按回车重新输入，**MATLAB** 会修正输出。

文件基础

下面我们以文件基本操作内容作为本章的结束。如果不能保存并重新取得我们的工作进度，**MATLAB** 就没有那么好用了，是吗？假定我们要保存在命令窗口中输入的变量和表达式以便后面使用，那么按照下面的步骤来操作就可以了：

1. 点击“文件（File）”下接菜单
2. 选择“保存工作区为（Save Workspace As…）”
3. 输入文件名
4. 点击“保存（Save）”按钮

在 Windows 下，这种方法创建了一个扩展名为.MAT 的 **MATLAB** 文件，如果你以这种方式保存文件，你可以在另一台计算机上运行程序重新取得所有命令然后继续工作。有时候，特别是复杂工程，你不会总想坐在一个地方把所有的表达式全部输进去，可能就想把很长的一系列命令保存到一个文件中，然后仅在命令窗口输入一个简单命令就能执行。创建一个**脚本文件**（*script file*）就能这样做了。这种类型的文件被称为 **MATLAB** 程序，以.M 为扩展名保存。因此，我们也称为 M 文件。我们也可以创建全是**函数**（*function*）的 M 文件。

到目前为止，我们已经知道如何创建脚本文件，脚本文件内容是一系列 **MATLAB** 命令。我们创建一个简单的脚本文件，用它来计算对不同 x 的 e^x 值。 首先，打开 **MATLAB** 编辑器，或者

- 从文件（File）下拉菜单中点击新建（New）→M 文件（M-File）
- 或者单击屏幕顶部工具栏上的新建图标（）

现在输入下面几行:

```
% script file example1.m to compute exponential of a set of numbers
x = [1:2:3:4];
y = exp(x)
```

注意第一行以%开始，表示这是一行注释（*Comment*）。这一行介绍性文字会方便我们，**MATLAB** 会忽略。下一行创建一个数组（*array*）或者称数集。数组采用方括号[]表示，元素之间采用冒号（:）或分号（;）隔开。最后一行告诉 **MATLAB** 计算数组中每个元素的幂，或者说计算 e^1，e^2，e^3，e^4 的值。点击保存图标（）或者从文件下拉菜单中选择“另存为”保存文件，以 example1.m 为文件名保存到 **MATLAB** 的当前目录中。

现在回到 **MATLAB** 的命令窗口，输入 `example1`。如果前面没有弄错，你会看到下面的输出

```
>> example1
y =
    2.7183    7.3891   20.0855   54.5982
```

我们也可以使用 M 文件创建和储存数据。我们在文件编辑器创建一组表示温度的数值并把它保存到文件中。在文件编辑器中创建一组温度值

```
temps = [32,50,65,70,85]
```

把文件保存到 **MATLAB** 当前目录并命名为 TemperatureData.m，只需在命令窗口中输入文件名即可访问刚才保存的文件，**MATLAB** 会把数分开后输出。

```
>> TemperatureData
temps =
    32    50    65    70    85
```

现在我们就可以引用文件中的数组名使用刚才的数据了。让我们创建一组表示摄氏温度的数字，然后转换成华氏温度，用下面的命令就可以了

```
>> CelsiusTemps = (5/9) * (temps - 32)
CelsiusTemps =
         0   10.0000   18.3333   21.1111   29.4444
```



在后面我们会学习 **MATLAB** 编程，到时会学习如何创建可以在命令窗口中调用的函数。

结束 MATLAB

到目前我们已经学习一些基础的 **MATLAB** 命令，你可能想保存工作然后退出 **MATLAB**。如何退出屏幕呢？与其它程序一样，你可以从文件下拉菜单中选择“退出（Exit）**MATLAB**”来结束 **MATLAB** 会话，还可以在命令窗口中输入 `quit` 命令或 `exit` 命令，这样也能关闭 **MATLAB**。

习题

用 **MATLAB** 计算下面各式：

1.$5\frac{11}{14}$

2.$5\frac{8}{3} + 3^7$

3.$9^{1.25}$

4.真或假。如果 y 尚未赋值，**MATLAB** 允许你定义方程 `x = y^2` 并储存到内存中，以后还可以使用它。

5.如果圆柱体的体积由其高 h 和半径 r 通过公式 $V=\pi r^2 h$ 得到，使用 **MATLAB** 求出高 12cm，直径 4cm 的圆柱体所包含的体积。

6.使用 **MATLAB** 计算 $\pi/3$ 的正弦值并用有理数（**一个能用整数或整数之比表达的数字，但零不能做分母。——译者注**）表示出来。

7.创建一个 M 文件，以有理数的形式显示 $\sin(\pi/4)$，$\sin(\pi/3)$ 和 $\sin(\pi/2)$ 的值。

【参考答案在第 226 页】

第二章



向量与矩阵

MATLAB 相当有用的其中一个领域是解决线性代数问题。这是因为 **MATLAB** 对处理数组有非凡能力，使得成为许多科学与工程应用中的一个有用的工具。



向量

向量（*vector*）一维数值数组。**MATLAB** 允许你创建列向量和行向量，列向量通过在方括号内把数值用分号（;）隔开来创建，对元素的个数没有限制。例如，要创建一个含有三个元素的列向量，我们写成：

```
>> a = [2; 1; 4]
a =
     2
     1
     4
```

列向量的基本操作通过引用创建时使用的变量名来进行的。如果我们要把一个列向量乘上一个数，这叫做**数量乘法**（*scalar multiplication*）。假设我们要创建一个新向量，它的元素是刚才我们创建的向量 a 的元素的三倍，我们可以先定义一个数量（注意命令行末的分号禁止了输出）：

```
>> c = 3;
```

下一步，我们就像 *a* 是**另一个变量**一样进行计算

```
>> b = c * a
b =
     6
     3
    12
```

要创建行向量，我们仍然是把一组数值用方括号括起来，不过这次使用的分隔符是空格（space）或逗号（,）。例如：

```
>> v = [2 0 4]
v =
     2     0     4
```

或者使用逗号：

```
>> w = [1,1,9]
w =
     1     1     9
```

使用转置操作可以进行列向量与行向量之间的相互转换。假设我们有 n 个元素的列向量，如下所示：

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ v_n \end{bmatrix}$$

转置后为

$$v^T=[v_1\ v_2 \ldots\ v_n]$$

在 **MATLAB** 中，我们用单引号（'）代表转置操作，把列向量转为行向量的例子为：

```
>> a = [2; 1; 4];
>> y = a'
y =
     2     1     4
```

现在我们把行向量转为列向量：

```
>> Q = [2 1 3]
Q =
     2     1     3
>> R = Q'
R =
     2
     1
     3
```

有时也可能会使用两个向量进行相加或相减来创建第三个。要实现此操作，两个向量之间必须类型相同，长度相同。因此我们可以对两个列向量进行相加创建第三个列向量，或者把两个行向量进行相加创建第三个行向量。此时只需引用变量名就可以了，没有必要列出所有的元素。下面的例子把两个列向量加到一起：

```
>> A = [1; 4; 5];
>> B = [2; 3; 3];
>> C = A + B
C =
     3
     7
     8
```

现在我让两个行向量相减：

```
>> W = [3,0,3];
>> X = [2,1,1];
>> Y = W - X
Y =
     1    -1     2
```


从已存变量创建大向量

MATLAB 允许把向量合并在一起创建新向量。设 **u** 和 **v** 是 **MATLAB** 中已经存在两个列向量，各自带有 *m* 和 *n* 个元素。我们创建第三个向量 **w**，它的前 *m* 个元素来自 **u**，后 *n* 个元素来自 **v**。新创建的列向量一共有 *m*+*n* 个元素。这时可以写成 **w** = [**u**; **v**]，例如:

```
>> A = [1; 4; 5];
>> B = [2; 3; 3];
>> D = [A;B]
D =
     1
     4
```

```
       5
       2
       3
       3
```

也可能使用行向量来创建新向量。要从带有 m 个元素的行向量 **r** 和带有 n 个元素的行向量 **s** 中创建带有 $m+n$ 个元素的行向量 **u**，我们写成 **u** = [**r**, **s**]。例如：

```
>> R = [12, 11, 9];
>> S = [1, 4];
>> T = [R, S]
T =
    12    11     9     1     4
```

创建等差元素向量

有时需要创建带有等差元素的向量，差值为 q 为一个实数。创建一个首元素为 x_i，末元素为 x_e 的向量 $\boldsymbol{x}$ 的语法如下：

$$\boldsymbol{x} = [x_i : q: x_e]$$

例如，要创建一个含有从 0 到 10 之间偶数的向量的写法为：

```
>> x = [0:2:10]
x =
     0     2     4     6     8    10
```

我们已经在前一章使用这种技术创建了一组数值。让我们看看在含有少量元素向量的背后它是如何工作的。开始我们创建一组值：

```
>> x = [0:0.1:1]
x =
  Columns 1 through 6
         0    0.1000    0.2000    0.3000    0.4000    0.5000
  Columns 7 through 11
    0.6000    0.7000    0.8000    0.9000    1.0000
```

这组值可以用来创建某函数在某点的一系列值。例如，假设 $y = e^x$，那么我们就有：

```
>> y = exp(x)
y =
  Columns 1 through 6
    1.0000    1.1052    1.2214    1.3499    1.4918    1.6487
  Columns 7 through 11
    1.8221    2.0138    2.2255    2.4596    2.7183
```

当然我们也可以让 $y = x^2$

```
>> y = x .^ 2
y =
  Columns 1 through 6
         0    0.0100    0.0400    0.0900    0.1600    0.2500
  Columns 7 through 11
```

```
    0.3600    0.4900    0.6400    0.8100    1.0000
```

插入语——注意在 **MATLAB** 中向量的乘方必须在幂运算符（^）前加上句号（.），如果我们只是输入 y = x^2，**MATLAB** 会给出错误信息。

```
>> y = x ^ 2
??? Error using ==> mpower
Matrix must be square.
```

回到创建等差元素数组的过程，注意你也可以使用负的递增量（即递减）。例如，我们创建一个从 100 到 80 以 5 增减的数列：

```
>> u = [100:-5:80]
u =
   100    95    90    85    80
```

我们也可以采用 *linspace* 命令创建行向量，这向量含有 *a* 到 *b* 之间间隔相等（等差）的 *n* 个元素。*linspace(a, b)* 创建了 *a*、*b* 之间含有 100 个等差元素的向量，而 *linspace(a, b, n)* 创建了 *a*、*b* 之间含有 *n* 个等差元素的向量。不管是哪种形式，**MATLAB** 自动确定元素之间的增量。

MATLAB 还允许创建 *n* 个**对数值**相隔相同的行向量，使用的格式为

logspace(a, b, n)

这创建了 10^a 和 10^b 之间 *n* 个对数值等差的向量。例如：

```
>> logspace(1,3,3)
ans =
          10         100        1000
```

另一个例子：

```
>> logspace(-1, 1, 6)
ans =
    0.1000    0.2512    0.6310    1.5849    3.9811   10.0000
```

特征化向量（Characterizing a Vector）

命令 *length* 返回向量中包含元素的个数，例如：

```
>> A = [2;3;3;4;5];
>> length(A)
ans =
     5

>> B = [1;1];
>> length(B)
ans =
     2
```

length 命令既可以应用到行向量和列向量（见本章后面的“矩阵基本操作”）也能应用到矩阵。

我们还可以使用 *max* 或 *min* 命令找出向量中数值最大和最小的元素。例如：



```
>> A = [8 4 4 1 7 11 2 0];
>> max(A)
ans =
    11

>> min(A)
ans =
     0
```

要找出向量的模，我们引进两种操作。回顾一下，向量

$$v = \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{bmatrix}$$

的模由下式给出：

$$|v| = \sqrt{v_1^2+v_2^2+...+v_n^2}$$

要实现本操作，我们先介绍向量的数量积（点乘），使用数组乘法（.*）来完成。首先我们定义一个向量。

```
>> J = [0; 3; 4];
```

现在我们就可以做数组相乘了：

```
>> J.*J
ans =
     0
     9
    16
```

本操作产生了元素为 v_1^2，v_2^2，...的向量。要得到我们要的总和可以使用 *sum* 操作符：

```
>> a = sum(J.*J)
a =
    25
```

向量的模为得数的平方根：

```
>> mag = sqrt(a)
mag =
     5
```

如果变量包含有复数，那么计算向量的模就要更加注意了。要计算复数行向量的模，必须先计算该向量的共轭复数向量。例如，如果：

$$u = \begin{bmatrix} i \\ 1+2i \\ 4 \end{bmatrix}$$

要计算模，我们需要下列的向量：

$$u^{\dagger}=[-i\ 1-2i\ 4]$$

那么我们需要计算的和是：

$$u^{\dagger}u =[-i\ 1-2i\ 4]\begin{bmatrix} i \\ 1+2i \\ 4 \end{bmatrix} =(-i)(i) + (1 - 2i)(1 + 2i) + (4)(4) = 22$$

因此这个复数向量的模是：

$$|u| = \sqrt{u^{\dagger}u} = \sqrt{22}$$

可以看到如何使用共轭复数计算向量的和，注意向量的模一定是实数。现在我们看看如何在 **MATLAB** 计算。首先输入列向量：

```
>> u = [i; 1+2i; 4];
```

如果我们以前例子的乘法计算向量的和，我们会得到不能正常工作的复数：

```
>> sum(u.*u)
ans =
  12.0000 + 4.0000i
```

因此我们先定义 *u* 的共轭复数，使用转置操作符，**MATLAB** 会自动计算：

```
>> v = u'
v =
        0 - 1.0000i   1.0000 - 2.0000i   4.0000
```

现在计算和：

```
>> b = sum(v.*u)
??? Error using ==> times
Matrix dimensions must agree.
```

非常不幸，看来我们是闯进死胡同，没有一一对应，如何绕过这一点呢？让我们计算复数的共轭复数向量，再来计算总和，我们使用 `conj` 命令计算向量的共轭复数向量：

```
>> v = conj(u)
v =
        0 - 1.0000i
   1.0000 - 2.0000i
   4.0000
```

现在我们得到正确答案，也就能正确得到模：

```
>> b = sum(v.*u)
b =
    22

>> magu = sqrt(b)
magu =
    4.6904
```

当然也可以一步到位：

```
>> c = sqrt(sum(conj(u).*u))
c =
    4.6904
```

其实在这里我们是以繁琐的方法来做的——只是为了演示方法和一些 **MATLAB** 命令，在下一节中，我们会讲到如何自动计算向量的模。

我们可以使用 `abs` 命令返回向量的绝对值——即在原位置返回向量中每个元素的绝对值。例如：



```
>> A = [-2 0 -1 9]
A =
   -2     0    -1     9
>> B = abs(A)
B =
    2     0     1     9
```

向量的点乘和叉乘（数量积和向量积）

两个向量 ***A***=（$a_1\ a_2\ \ldots\ a_n$）和 ***B***=（$b_1\ b_2\ \ldots\ b_n$）的**点乘**结果由下式给出

$$\boldsymbol{A}\cdot\boldsymbol{B}=\sum_i a_i b_i$$

在 **MATLAB** 中，***a***、***b*** 两向量的点乘可以使用 dot(***a***, ***b***)命令计算。

两个向量点乘的结果是数量，也即是说，它只是一个数值。我们使用 **MATLAB** 计算一个简单的例子：

```
>> a = [1;4;7]; b = [2;-1;5];
>> c = dot(a,b)
c =
   33
```

点乘可以用来计算向量的模，所需要的只是把向量同时传递给 *dot* 命令的两个参数。考虑上一节的向量：

```
>> J = [0; 3; 4];
```

调用 dot 命令我们得到：

```
>> dot(J, J)
ans =
   25
```

我们也可以用下面这种方式计算向量的模：

```
>> mag = sqrt(dot(J,J))
mag =
    5
```

对于带有复数元素的向量，dot 操作也能正确计算：

```
>> u = [-i; 1 + i; 4 + 4*i];
>> dot(u, u)
ans =
   35
```

向量的另一个重要操作是**叉乘**。要计算向量的叉乘，这两个向量必须是的三维的。例如：

```
>> A = [1 2 3]; B = [2 3 4];
>> C = cross(A, B)
```

```
C =
    -1     2    -1
```

引用向量元素

MATLAB 有几种方法用来引用向量的一个或多个元素。向量 **v** 的第 *i* 个元素可以用 **v**(*i*)来引用，例如：

```
>> A = [12; 17; -2; 0; 4; 4; 11; 19; 27];
>> A(2)
ans =
    17
>> A(8)
ans =
    19
```

如果使用冒号——如 **v**(:)——来引用向量，等于告诉 **MATLAB** 列出向量的所有元素：

```
>> A(:)
ans =
    12
    17
    -2
     0
     4
     4
    11
    19
    27
```

我们还可以选出向量中某一范围内的元素，本节中我们一直在使用的向量 ***A*** 有 9 个元素，可以用 ***A***(4:6)选出第 4 到第 6 个元素组成一个新的、含有 3 个元素的向量：

```
>> v = A(4:6)
v =
     0
     4
     4
```

在下一节中，我们会看到如何使用这项技术来引用被称为**矩阵**的整行或整列。

矩阵基本操作

矩阵是两维数字数组，要在 **MATLAB** 创建矩阵，输入的行各元素之间用空格或逗号分隔，行末使用分号标记。例如，考虑下列例子：

$$\mathbf{A} = \begin{bmatrix} -1 & 6 \\ 7 & 11 \end{bmatrix}$$

这个矩阵在 **MATLAB** 中使用下面的语法输入：



```
>> A = [-1,6; 7, 11]
A =
    -1     6
     7    11
```

如果考虑矩阵

$$B = \begin{bmatrix} 2 & 0 & 1 \\ -1 & 7 & 4 \\ 3 & 0 & 1 \end{bmatrix}$$

在 **MATLAB** 中我们以下面的方式输入：

```
>> B = [2,0,1;-1,7,4; 3,0,1]
B =
     2     0     1
    -1     7     4
     3     0     1
```

我们已经在向量中使用的很多操作也可以延伸到矩阵操作。所有的带有 n 个元素的列向量是一个有一列 n 行的矩阵，所有的带有 n 个元素的行向量是一个有一行 n 列的矩阵。例如，数乘可以通过引用矩阵的名称来进行：

```
>> A = [-2 2; 4 1]
A =
    -2     2
     4     1
>> C = 2*A
C =
    -4     4
     8     2
```

如果两个矩阵行数和列数都相等，那么它们可以进行相加减：

```
>> A = [5 1; 0 9];
>> B = [2 -2; 1 1];
>> A + B
ans =
     7    -1
     1    10
>> A - B
ans =
     3     3
    -1     8
```

我们也可以进行矩阵的转置。转置操作交换矩阵的行和列。例如：

$$A = \begin{bmatrix} -1 & 2 & 4 \\ 0 & 1 & 6 \\ 2 & 7 & 1 \end{bmatrix}, \Rightarrow A^T = \begin{bmatrix} -1 & 0 & 2 \\ 2 & 1 & 7 \\ 4 & 6 & 1 \end{bmatrix}$$

我们使用跟转置向量相同的操作符转置矩阵：

```
>> A = [-1 2 0; 6 4 1]
A =
    -1     2     0
     6     4     1
>> B = A'
B =
```

```
     -1     6
      2     4
      0     1
```

如果矩阵包含有复数元素，那么转置操作会自动计算复数的共轭值：

```
>> C = [1+i, 4-i; 5+2*i, 3-3*i]
C =
   1.0000 + 1.0000i   4.0000 - 1.0000i
   5.0000 + 2.0000i   3.0000 - 3.0000i
>> D = C'
D =
   1.0000 - 1.0000i   5.0000 - 2.0000i
   4.0000 + 1.0000i   3.0000 + 3.0000i
```

如果要转置复数矩阵的而不计算它的共轭值，那么我们使用（.'）：

```
>> D = C.'
D =
   1.0000 + 1.0000i   5.0000 + 2.0000i
   4.0000 - 1.0000i   3.0000 - 3.0000i
```

我们可以进行**数组**相乘，注意这**不是矩阵相乘**。我们使用与向量相乘相同的符号（.*）。例如：

```
>> A = [12 3; -1 6]; B = [4 2; 9 1];
>> C = A .* B
C =
     48     6
     -9     6
```

正如所见。A .* B 操作实际上是对应位置元素与元素进行相乘。理论上它计算如下：

$$\boldsymbol{A} .* \boldsymbol{B} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} (a_{11})(b_{11}) & (a_{12})(b_{12}) \\ (a_{21})(b_{21}) & (a_{22})(b_{22}) \end{pmatrix}$$

下一节我们会学习如何进行矩阵相乘。

矩阵相乘

考虑两个矩阵 ***A*** 和 ***B***，如果 ***A*** 是一个 $m \times p$ 矩阵，而 ***B*** 是 $p \times n$ 矩阵，它们可以相乘产生 $m \times n$ 矩阵。要在 **MATLAB** 中这样做，我们只需把点号去掉简单地写成 ***A*B*** 即可。要紧紧记住：如果两个矩阵的维数不正确，那么会产生错误。考虑下面两个矩阵：

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 3 & 4 \\ 5 & 6 \end{pmatrix}$$

他们都是 2×2 矩阵，因此可以进行矩阵相乘。首先我们进行数组相乘以便看出差异：

```
>> A = [2 1; 1 2]; B = [3 4; 5 6];
>> A .* B      %数组相乘
ans =
      6     4
      5    12
```

现在我们把“.”号去掉，进行结果不同的矩阵相乘：



```
>> A * B      %矩阵相乘
ans =
    11    14
    13    16
```

下面是另一个例子，考虑：

$$A=\begin{pmatrix}1 & 4\\ 8 & 0\\ -1 & 3\end{pmatrix},\ B=\begin{pmatrix}-1 & 7 & 4\\ 2 & 1 & -2\end{pmatrix}$$

矩阵 ***A*** 是一个 3×2 矩阵，而 ***B*** 是一个 2×3 矩阵，由于 ***A*** 的列数和 ***B*** 的行数相匹配。我们可以计算 ***AB*** 的向量积。在 **MATLAB** 中：

```
>> A = [1 4; 8 0; -1 3]; B = [-1 7 4; 2 1 -2];
>> C = A*B
C =
     7    11    -4
    -8    56    32
     7    -4   -10
```

虽然这种形式的矩阵可以进行相乘，但不能进行数组相乘。要使用数组相乘，行数和列数都必须相匹配。这时如果进行数组相乘，**MATLAB** 会告诉我们：

```
>> C = A .* B
??? Error using ==> times
Matrix dimensions must agree.
```

更多基本操作

MATLAB 允许在矩阵上进行其它几个操作，对这几个操作，从你的线性代数背景来说，你可能不能立即习惯它们。例如，**MATLAB** 允许你把数量添加到一个数组（向量或矩阵）中，即把数加到数组的每个元素中。下面是把数加到行向量的一个例子：

```
>> A = [1 2 3 4];
>> b = 2;
>> C = b + A
C =
     3     4     5     6
```

我们也可以在数组上进行左除和右除。这时数组元素与元素匹配相除，因此两数组必须等大。例如，我们用“./”让 **MATLAB** 进行数组右除：

```
>> A = [2 4 6 8]; B = [2 2 3 1];
>> C = A ./ B
C =
     1     2     2     8
```

数组左除用 `C = A .\ B` （此式与 `C = B ./ A` 相同）简要说明：

```
>> C = A .\ B
C =
```

```
      1.0000    0.5000    0.5000    0.1250
```

基本上你能想到的任何数学操作在 **MATLAB** 中都能在数组中实现。例如，我们可以对每个元素进行平方：

```
>> B = [2 4; -1 6]
B =
     2     4
    -1     6
>> B .^ 2
ans =
     4    16
     1    36
```

特殊类型矩阵

单位矩阵是主对角线元素全为 1 其它元素全为 0 的方形矩阵。要创建 $n{\times}n$ 的单位矩阵，输入下面的 **MATLAB** 命令：

eye(n)

我们创建 4×4 单位矩阵：

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

要创建 $n{\times}n$ 的零矩阵，我们输入 *zeros*(*n*)。我们还可以输入 *zeros*(*m*, *n*)创建 $m{\times}n$ 的矩阵，当然也完全可以创建整个元素都为 1 的矩阵。可能出乎你意外，你只需输入 *ones*(*n*)或 *ones*(*m*,*n*)即可分别创建 $n{\times}n$ 和 $m{\times}n$ 的矩阵。

引用矩阵元素

在 **MATLAB** 中，矩阵的单个元素或整列都能够被引用。看下面的矩阵：

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
```

我们可以用 ***A***(*m*,*n*)选出第 *m* 行 *n* 列的元素。例如：

```
>> A(2,3)
ans =
     6
```

要引用第 *i* 列的所有元素，输入 ***A***(:,*i*)。例如，我们要选出第二列的所有元素：

```
>> A(:,2)
ans =
     2
     5
     8
```

要选出从第 *i* 列到第 *j* 列之间的所有元素，我们输入 ***A***(:,*i*:*j*)。下面的例子返回第二和第三列的元素：

```
>> A(:,2:3)
ans =
     2     3
     5     6
     8     9
```

我们也可以选出小块或子矩阵。仍然用刚才的矩阵，我们选出第二到第三行同时处于第一和第二列的元素，写成：

```
>> A(2:3,1:2)
ans =
     4     5
     7     8
```

我们也可以通过这些引用改变矩阵的值。让我们把第一行第一列元素的值改为-8：

```
>> A(1,1) = -8
A =
    -8     2     3
     4     5     6
     7     8     9
```

要在 **MATLAB** 中创建空数组，只需在方括号[]里留空即可。它可以用来删除矩阵的行或列。让我们删除 ***A*** 的第二行：

```
>> A(2,:)=[]
A =
    -8     2     3
     7     8     9
```

本操作把前面 3×3 矩阵变成 2×3 矩阵。当然也可以通过引用矩阵中的行或列来创建新的矩阵。在本例中，我们复制 ***A*** 矩阵的第一行四次来创建一个新矩阵：

```
>> E=A([1,1,1,1],:)
E =
    -8     2     3
    -8     2     3
    -8     2     3
    -8     2     3
```

下面这个例子引用两次 ***A*** 的第一行创建新矩阵：

```
>> A = [-8,2,3;7,8,9];
>> F = A([1,2,1],:)
F =
```

```
      -8     2     3
       7     8     9
      -8     2     3
```

行列式与线性方程组求解

方矩阵的行列式是一个数值。对于一个 2×2 矩阵，其行列式由正式给出：

$$\boldsymbol{D} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

要在 **MATLAB** 中计算矩阵 ***A*** 的行列式，简单地写成 det(***A***)即可，下面是 2×2 矩阵的行列式：

```
>> A = [1 3; 4 5];
>> det(A)
ans =
    -7
```

在下面这个例子中，我们得到 4×4 矩阵的行列式：

```
>> B = [3 -1 2 4; 0 2 1 8; -9 17 11 3; 1 2 3 -3];
>> det(B)
ans =
  -533
```

不用说，这种计算用手工完成是相当单调乏味的。行列式可以用来找出一个线性方程组是否有解。考虑下面的方程组：

$$5x + 2y - 9z = -18$$
$$-9x - 2y + 2z = -7$$
$$6x + 7y + 3z = 29$$

要找出像这样的方程组的解，需要两步：首先系数矩阵 ***A*** 的行列式，在本例中是：

$$\boldsymbol{A} = \begin{pmatrix} 5 & 2 & -9 \\ -9 & -2 & 2 \\ 6 & 7 & 3 \end{pmatrix}$$

它的行列式是：

```
>> A = [5 2 -9; -9 -2 2; 6 7 3]
A =
     5     2    -9
    -9    -2     2
     6     7     3
>> det(A)
ans =
   437
```

如果行列式不为零，那么解存在。解是列向量：

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

在 **MATLAB** 中我们使用左除就可很容易得到解。首先，我们创建由方程组右边组成的向量。得到：

```
>> b = [-18;-7;29];
```

```
>> A \ b
ans =
    1.0000
    2.0000
    3.0000
```

求矩阵的秩

矩阵的**秩**是矩阵行或列的数值**线性独立**的度量。如果一个向量线性独立于另外一些向量组，那意味着这一个向量不能写成它们的线性组合。简单例子如下：

$$\boldsymbol{u}=\begin{pmatrix}1\\-2\end{pmatrix},\ \boldsymbol{v}=\begin{pmatrix}3\\-4\end{pmatrix},\ \boldsymbol{w}=\begin{pmatrix}5\\-6\end{pmatrix}$$

研究这些列向量，我们可以看出：

$$2\boldsymbol{u}+\boldsymbol{v}=\boldsymbol{w}$$

因此 $\boldsymbol{w}$ 线性相关于 $\boldsymbol{u}$ 和 $\boldsymbol{v}$，此时 $\boldsymbol{w}$ 可以写成 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 的线性组合。另一例子：

$$\boldsymbol{u}=\begin{pmatrix}2\\0\\0\end{pmatrix},\ \boldsymbol{v}=\begin{pmatrix}0\\-1\\0\end{pmatrix},\ \boldsymbol{w}=\begin{pmatrix}0\\0\\7\end{pmatrix}$$

就构成线性独立组，这是因为这些向量中没有一个可以写成另外两个的线性组合。

再看矩阵：

$$\boldsymbol{A}=\begin{pmatrix}0&1&0&2\\0&2&0&4\end{pmatrix}$$

很明显，矩阵的第二行是第一行的两倍。因此只有一行是独立的，矩阵的秩为 1 。我们在 **MATLAB** 用下面的方法计算秩检验一下：

```
>> A = [0 1 0 2; 0 2 0 4];
>> rank(A)
ans =
     1
```

再看一个例子：

$$\boldsymbol{B}=\begin{pmatrix}1&2&3\\3&0&9\\-1&2&-3\end{pmatrix}$$

第三列是第一列的三倍：

$$\begin{pmatrix}3\\9\\-3\end{pmatrix}=3\begin{pmatrix}1\\3\\-1\end{pmatrix}$$

因此，这两列线性相关，而另外两列线性独立，这是因为找不到常量 α 使得：

$$\begin{pmatrix}2\\0\\2\end{pmatrix}=\alpha\begin{pmatrix}1\\3\\-1\end{pmatrix}$$

到此我们知道有两个线性独立列，`rank(B) = 2`。在 **MATLAB** 检验一下：

```
>> B = [1 2 3; 3 0 9; -1 2 -3];
>> rank(B)
ans =
     2
```

现在我们看一下带有 n 个未知量的 m 个线性方程组：

$$\boldsymbol{Ax}=\boldsymbol{b}$$

把 $\boldsymbol{b}$ 连结 $\boldsymbol{A}$ 上构成了**增广矩阵**：

$$[\boldsymbol{A} \quad \boldsymbol{b}]$$

当且仅当 $rank(\boldsymbol{A}) = rank(\boldsymbol{A}\ \boldsymbol{b})$时方程组有解。如果秩等于 *n*，那么方程组有唯一解，但如果秩小于 *n*，那么方程组有无数解。如果用 *r* 表示秩，那么未知数 *r* 就可以表示成其它 *n-r* 个变量的线性组合。

既然秩可以在 **MATLAB** 中很受容易得到，也能够方便地把数组连在一起，我们就可以相对轻松地分析线性方程组了。如果秩条件满足秩与未知数个数相等，解就可以用左除计算得到。我们演示一下这个技术。看方程组：

```
 x - 2y + z = 12
3x + 4y +5z = 20
-2x +  y +7z = 11
```

矩阵的系数是：

$$\boldsymbol{A} = \begin{pmatrix} 1 & -2 & 1 \\ 3 & 4 & 5 \\ -2 & 1 & 7 \end{pmatrix}$$

我们还有：

$$\boldsymbol{b} = \begin{pmatrix} 12 \\ 20 \\ 11 \end{pmatrix}$$

因此增广矩阵是：

$$(\boldsymbol{A}\ \boldsymbol{b}) = \begin{pmatrix} 1 & -2 & 1 & 12 \\ 3 & 4 & 5 & 20 \\ -2 & 1 & 7 & 11 \end{pmatrix}$$

第一步是在 **MATLAB** 中输入这些矩阵：

```
>> A = [1 -2 1; 3 4 5; -2 1 7]; b = [12; 20; 11];
```

接着我们创建增广矩阵：

```
>> C = [A b]
C =
     1    -2     1    12
     3     4     5    20
    -2     1     7    11
```

现在我们检查一下 ***A*** 的秩：

```
>> rank(A)
ans =
     3
```

增广矩阵的秩为：

```
>> rank(C)
ans =
     3
```

由于秩相同，因此解存在。这里有三个未知量，我们也注意到秩 *r* 满足 $r = n$。这意味着解唯一。我们用左除求得解：

```
>> x = A \ b
x =
    4.3958
   -2.2292
    3.1458
```



求逆矩阵与伪逆矩阵

矩阵 $\boldsymbol{A}$ 的**逆矩阵**用 $\boldsymbol{A}^{-1}$ 表示，并且满足下面的关系：

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{E}$$

看下面的矩阵方程：

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

如果 $\boldsymbol{A}$ 的逆矩阵存在，那么解可写成：

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

在实践中它要比看起来难得多，这是因为计算逆矩阵是相当繁琐的。幸运的是 **MATLAB** 帮我们免去这些繁琐工作，使得计算变得简单。在 **MATLAB** 中输入下面的命令即可计算矩阵 $\boldsymbol{A}$ 的逆矩阵：

inv($\boldsymbol{A}$)

逆矩阵并不一定存在。事实上我们可以用矩阵的行列式确定逆矩阵是否存在。如果 det($\boldsymbol{A}$) = 0，那么逆矩阵不存在，这时我们说此矩阵是一个**奇异矩阵**。

下面我们开始计算一些逆矩阵，体会一下在 **MATLAB** 中计算逆矩阵是多么简单的一件事。从一个 2×2 矩阵开始：

$$\boldsymbol{A} = \begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$$

首先检查矩阵的行列式：

```
>>  A = [2 3; 4 5]
A =
     2     3
     4     5
>> det(A)
ans =
    -2
```

由于 det($\boldsymbol{A}$) ≠ 0，我们可以求逆矩阵。**MATLAB** 告诉我们它是：

```
>> inv(A)
ans =
   -2.5000    1.5000
    2.0000   -1.0000
```

我们可以手工计算验证这个逆矩阵：

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}\begin{pmatrix} -5/2 & 3/2 \\ 2 & -1 \end{pmatrix} = \begin{pmatrix} -10/2 + 6 & 6/2 - 3 \\ -20/2 + 10 & 12/2 - 5 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

这里我并不是要告诉你如何手工计算逆矩阵——这些知识可以在很多线性代数书中找到。可以这么说，你倒是要避免这么做，特别是对于大矩阵。我们看看在 **MATLAB** 中 4×4 的例子。

首先创建矩阵：

```
>>  S = [1 0 -1 2; 4 -2 -3 1; 0 2 -1 1; 0 0 9 8];
```

检查行列式：

```
>> det(S)
ans =
  -108
```

由于 det($\boldsymbol{S}$) ≠ 0，逆矩阵存在。**MATLAB** 输出为：

```
>> T = inv(S)
T =
   -0.9259    0.4815    0.4815    0.1111
   -0.6296    0.1574    0.6574    0.0556
   -0.5926    0.1481    0.1481    0.1111
    0.6667   -0.1667   -0.1667         0
```

要手工计算它可就不是那么轻松愉快了，并且很有可能计算错。我们检查一下：

```
>> S*T
ans =
    1.0000         0         0         0
    0.0000    1.0000   -0.0000         0
    0.0000   -0.0000    1.0000         0
         0         0         0    1.0000
```

–0.0000 来自于四舍五入误差。在计算机的精度范围内，看来我们确实得到了逆矩阵。我们用另一种方式检查一下乘积：

```
>> T*S
ans =
     1.0000         0          0         0
          0    1.0000          0    0.0000
          0         0     1.0000         0
          0         0     0.0000    1.0000
```

这一次输出结果变得好看一点了（管它 **MATLAB** 内部如何处理，反正对我们这些基本应用影响不重要）。

现在我们看看如何使用逆矩阵求解方程组。考虑方程组：

$$3x - 2y = 5$$
$$6x - 2y = 2$$

系数矩阵是：

```
>> A = [3 -2; 6 -2]
A =
     3    -2
     6    -2
```

$\boldsymbol{Ax} = \boldsymbol{b}$ 中的向量 $\boldsymbol{b}$ 是：

```
>> b = [5;2]
b =
     5
     2
```

我们检查 $\boldsymbol{A}$ 的行列式确保逆矩阵存在：

```
>> det(A)
ans =
     6
```

既然逆矩阵存在，我们就可以在 **MATLAB** 很容易求得解：

```
>> x = inv(A) * b
```



```
x =
   -1.0000
   -4.0000
```

我们也可以仅使用前面讨论的方法，用系数矩阵的逆矩阵来相乘求得解，如果系数矩阵是方阵，这意味着方程组的方程数与未知数个数相同。如果方程数比未知数个数少，此时方程组称为**欠定**。这意味着方程组有无限解，因为此时只有一些未知数能够确定下来，不能确定的未知数可以赋予任意值，因此可以得到无限多组解。我们举个简单例子：

$$\begin{aligned} x + 2y - z &= 3 \\ 5y + z &= 0 \end{aligned}$$

稍微处理可以得到：

$$\begin{aligned} z &= -5y \\ x &= 3 - 7y \end{aligned}$$

在这个方程组中，我们可以为两个变量找到值（x 和 z），第三个变量 y 是不确定的。我们可以为 y 选择喜欢的值，此时方程组就有解了。

另外一种方程组存在无限解的情况是当 $\det(A) = 0$ 时。

那么，要是可怜的数学家遇到这种情况怎么办呢？幸运的是，**伪逆矩阵（属广义逆矩阵——译者注）**来救援。解的办法是为未知数给出最小范数实数解，即是说，解向量 x 具有最小范数且元素都为实数。我们找一个线性方程组来举例：

$$\begin{aligned} 3x + 2y - z &= 7 \\ 4y + z &= 2 \end{aligned}$$

很明显，这个方程组具无限解。我们输入数据：

```
>> A= [3 2 -1; 0 4 1]; b = [7; 2];
>> C = [A b]
C =
     3     2    -1     7
     0     4     1     2
```

现在计算秩：

```
>> rank(A)
ans =
     2
>> rank(C)
ans =
     2
```

由于这些秩相等，解存在。我们可以用 **MATLAB** 中的左除产生一组解：

```
>> x = A \ b
x =
    2.0000
    0.5000
         0
```

MATLAB 通过把其中一个变量（本例中是 z）设为零产生一组解。如果要尝试用左除产生一组解，通常都是这样做的。当然，这解是有效的，不过记住它只是让 $z = 0$，而 z 可以是任何值。

我们也可以使用伪逆矩阵来解这个方程组。输入如下内容：

```
>> x = pinv(A) * b
x =
   1.6667
```

```
     0.6667
    -0.6667
```

MATLAB 使用**摩尔-彭罗斯 pseudoinverse** 法计算 ***pinv***。

简化阶梯矩阵

MATLAB 中的 `rref(`*A*`)`函数使用高斯-乔丹（Gauss-Jordan）消元法产生矩阵 ***A*** 降阶后的阶梯形式。你可以用手工计算验证这个例子：

```
>> A = [1 2; 4 7]
A =
     1     2
     4     7
>> rref(A)
ans =
     1     0
     0     1
```

现在让我们看一个你可能不大会想用手工计算的例子。魔方矩阵（幻方）是一个 $n \times n$ 矩阵。矩阵的元素从 1 到 n^2之间，并且行元素的和等于列元素的和。如果要手工计算产生魔方那可能要搞到脑出血。不过可以用 **MATLAB** 的 `magic(n)`命令帮我们算出来。例如：

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

我们检查一下每列之和是否相等：

```
>> sum(A)
ans =
    65    65    65    65    65
```

使用高斯-乔丹消元法化简成阶梯形式，对我来说有些难，所以我打算让 **MATLAB** 告诉我们它是怎样的：

```
>> rref(A)
ans =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

怎样？我们得到一个单位矩阵。试试更大一点的矩阵：

```
>> magic(8)
```



```
ans =
    64     2     3    61    60     6     7    57
     9    55    54    12    13    51    50    16
    17    47    46    20    21    43    42    24
    40    26    27    37    36    30    31    33
    32    34    35    29    28    38    39    25
    41    23    22    44    45    19    18    48
    49    15    14    52    53    11    10    56
     8    58    59     5     4    62    63     1
```

这个矩阵对于大多数人来说难以处理，不过在 **MATLAB** 求得如下：

```
>> rref(magic(8))
ans =
     1     0     0     1     1     0     0     1
     0     1     0     3     4    -3    -4     7
     0     0     1    -3    -4     4     5    -7
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

矩阵分解

MATLAB 可以快速地对矩阵进行 LU(三角)分解、QR(正交三角) 分解和 SVD（奇异值）分解。在这一节中，我们看看如何在 **MATLAB** 中进行 LU 分解及如何使用它来求解线性方程组。对矩阵 ***A*** 进行 LU 分解写成：

```
[L, U] = lu(A)
```

例如，让我们找出下列矩阵的 LU 分解：

$$\boldsymbol{A} = \begin{pmatrix} -1 & 2 & 0 \\ 4 & 1 & 8 \\ 2 & 7 & 1 \end{pmatrix}$$

我们输入矩阵得到：

```
>> A = [-1 2 0; 4 1 8; 2 7 1];
>> [L, U] = lu(A)
L =
   -0.2500    0.3462    1.0000
    1.0000         0         0
    0.5000    1.0000         0
U =
    4.0000    1.0000    8.0000
         0    6.5000   -3.0000
         0         0    3.0385
```

我们可以使用 LU 分解求解线性方程组。假设 ***A*** 是某个方程组的系数矩阵，而

$$\boldsymbol{b} = \begin{pmatrix} 12 \\ -8 \\ 6 \end{pmatrix}$$

方程组的解可以通过两次左除得到：

```
x = U\(L\b)
```

我们求得：

```
>> b = [12;-8;6];
>> x = U\(L\b)
x =
   -6.9367
    2.5316
    2.1519
```

看方程组：

```
                    3x + 2y  - 9z  = -65
                   -9x + 5y  + 2z  = 16
                    6x + 7y  + 3z  = 5
```

我们把方程组输进 **MATLAB**：

```
>> A = [3 2 -9; -9 -5 2; 6 7 3]; b = [-65; 16; 5];
```

现在求出 ***A*** 的 LU 分解：

```
>> [L, U] = lu(A)
L =
   -0.3333    0.0909    1.0000
    1.0000         0         0
   -0.6667    1.0000         0
U =
   -9.0000   -5.0000    2.0000
         0    3.6667     4.3333
         0         0   -8.7273
```


现在我们使用这些矩阵用左除来求得解：

```
>> x = U\(L\b)
x =
    2.0000
   -4.0000
    7.0000
```

习题

1. 求向量 ***A*** = (-1 7 3 2)的模。
2. 求向量 ***A*** = (-1+*i* 7*i* 3 -2-2*i*)的模。
3. 考虑数 1，2，3。用这些数做元素分别以行向量或列向量的形式输入 **MATLAB**。
4. 设 ***A*** = [1; 2; 3]， ***B*** = [4; 5; 6]，求这两个向量的数组乘积。
5. 什么命令可以产生一个只有对角元素为 1，其它元素全为 0 的 5×5 矩阵。
6. 计算两个矩阵 $A = \begin{pmatrix} 8 & 7 & 11 \\ 6 & 5 & -1 \\ 0 & 2 & -8 \end{pmatrix}$，$B = \begin{pmatrix} 2 & 1 & 2 \\ -1 & 6 & 4 \\ 2 & 2 & 2 \end{pmatrix}$ 的数组乘积和矩阵乘积。
7. 设 $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$，使用它创建 $B = \begin{pmatrix} 7 & 8 & 9 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{pmatrix}$。
8. 求下面方程组的解：



$$\begin{aligned} x + 2y + 3z &= 12 \\ -4x + y + 2z &= 13 \\ 9y - 8z &= -1 \end{aligned}$$

系数矩阵的行列式等于多少？

9. 下面方程组的解存在吗？它是什么？

$$\begin{aligned} x - 2y + 3z &= 1 \\ x + 4y + 3z &= 2 \\ 2x + 8y + z &= 3 \end{aligned}$$

10. 使用 LU 分解求解方程组：

$$\begin{aligned} x + 7y - 9z &= 12 \\ 2x - y + 4z &= 16 \\ x + y - 7z &= 16 \end{aligned}$$

【参考答案在第 226 页】

第三章



绘图与图形

绘图是数学应用程序在计算机上最有用的一种应用，**MATLAB** 当然也毫不例外。有时我们需要将使手工难以绘制的函数或实验数据的可视化。本章我们将介绍在 **MATLAB** 中完成这些任务的命令和技术。



2D 绘图基础

我们从绘制最基本的图形开始——只有一个变量的函数图形。在 **MATLAB** 中绘图包含下面三个步骤：

1. 定义函数
2. 指定要绘制的函数图形的值范围
3. 调用 **MATLAB** 的 `plot(x, y)`函数

当指定函数值的范围时，我们必须告诉 **MATLAB** 函数使用的变量增量。使用较少的增量可以使得图形显示更加平滑。如果增量较小，**MATLAB** 会计算更多的函数值，不过通常不需要取得那么小。我们用一个简单的例子来看看如何做。

我们绘制 $0 \leqslant x \leqslant 10$ 之间的 $y = \cos(x)$的图形。绘制之前，我们要定义这个区间并告诉 **MATLAB** 我们所使用的增量。区间使用方括号[]括起来，以下面的形式定义：

```
[ start : interval : end ]
```

例如，如果我们要告诉 **MATLAB** 在 $0 \leqslant x \leqslant 10$ 上以 0.1 的增量递增，我们输入：

```
[0:0.1:10]
```

我们用赋值运算符给这个范围内的变量一个名称，也用这种办法告知 **MATLAB** 相关变量和我们要绘制的函数。因此，要绘制 $y = \cos(x)$，输入的命令如下：

```
>> x = [0:0.1:10];
>> y = cos(x);
```

注意我们每行都以分号“;”结尾，记住，这会抑制 **MATLAB** 输出。你不会想让 **MATLAB** 在屏幕中间输出一大串 x 值，因此使用了分号。现在我们可以输入下面的命令绘图了：

```
>> plot(x, y)
```

输入绘图命令后敲回车 ENTER。过一会儿，**MATLAB** 会新开启一个标题为“Figure 1”的新窗口，窗口中含有所绘制的图形。本例中我们得到图 3-1。

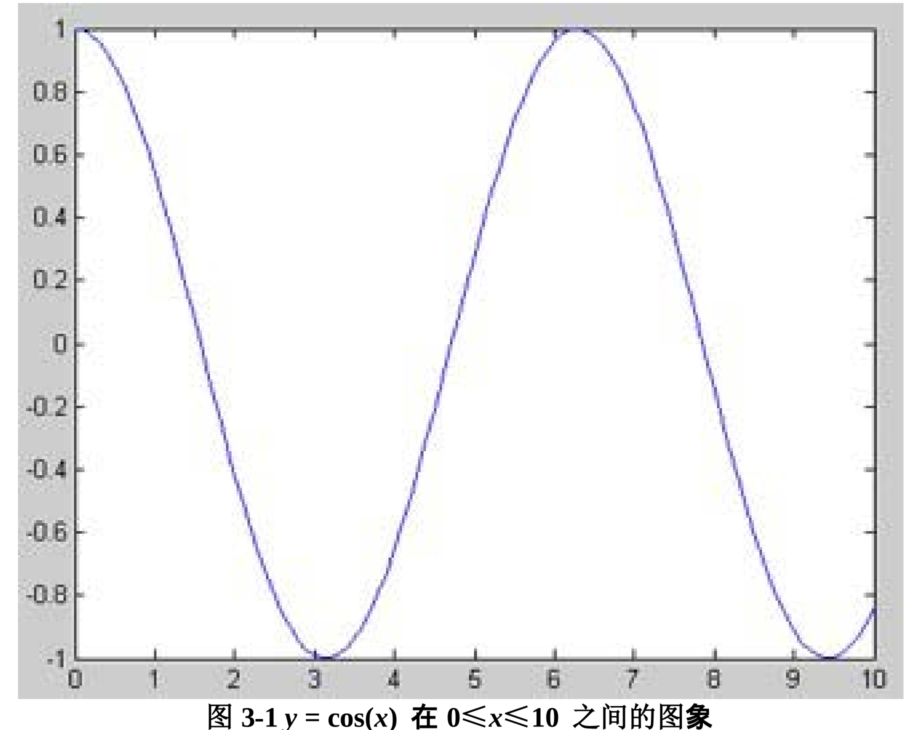


图 3-1 $y = \cos(x)$ 在 $0 \leqslant x \leqslant 10$ 之间的图象

现在增量呢？假设我们把增量扩大 10 倍，即把它设置为 1，此时使用输入下面的命令

即可：

```
>> x = [0:1:10];
```

此时如果尝试再次绘图，我们会得到错误信息：

```
>> plot(x, y);
??? Error using ==> plot
Vectors must be the same lengths.
```

我们已经定义过 $y = \cos(x)$，因此 **MATLAB** 不能再次绘图。那怎么办？我们必须告诉 **MATLAB** 重新计算我们新定义 x 后的 y 值。换句话说，正确的行为是我们重新输入所有的命令：

```
>> y = cos(x)
y =
  Columns 1 through 5
    1.0000    0.5403   -0.4161   -0.9900   -0.6536
  Columns 6 through 10
    0.2837    0.9602    0.7539   -0.1455   -0.9111
  Column 11
   -0.8391
>> plot(x, y)
```

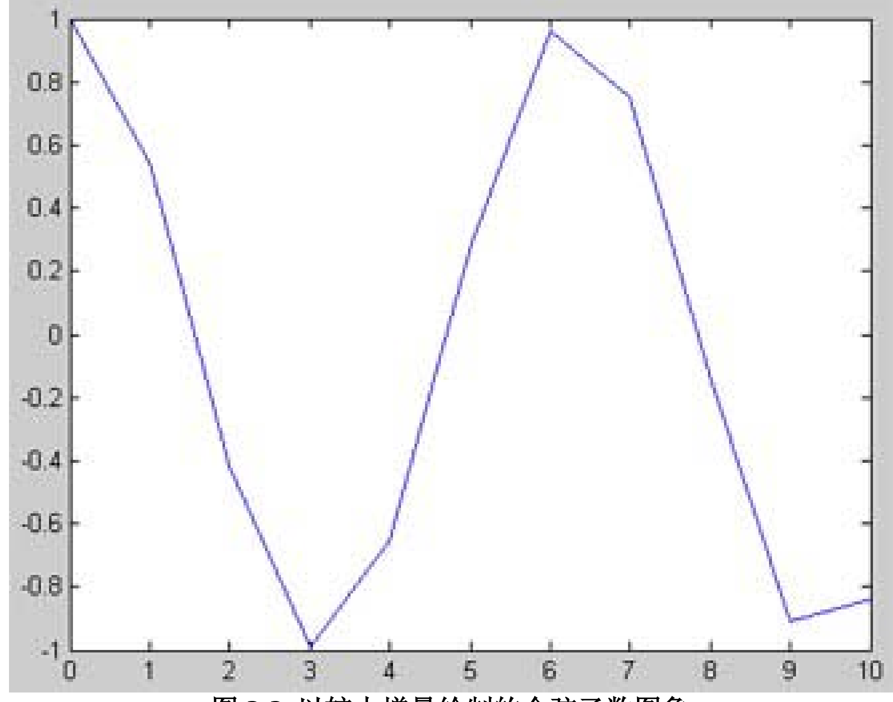


图 3-2 以较大增量绘制的余弦函数图象

简短插入语——注意我们在重新定义 y 的时候去掉了末尾的分号，所以 **MATLAB** 输出了每个 x 点的 $cos(x)$ 值。此时如果 x 点的数量巨大，你会觉得这并不需要显示出来。

好了，现在回到绘图。当我们以较大增量绘图时，所绘制的图象就不那么精确了。看看图 3-2 中 **MATLAB** 以 1 为增量绘制的图象，此时变得很粗劣。让我们用另一种方法试试。我们把增量设为原来的 1/10，即设为 0.01。

记住我们需要重新定义 y，因此我们需要输入的命令是：

```
>> x = [0:0.01:10];
>> y = cos(x);
```



```
>> plot(x, y)
```

这一次我们重现了非常漂亮的 $y = \cos(x)$ 图象，如图 3-3 所示。

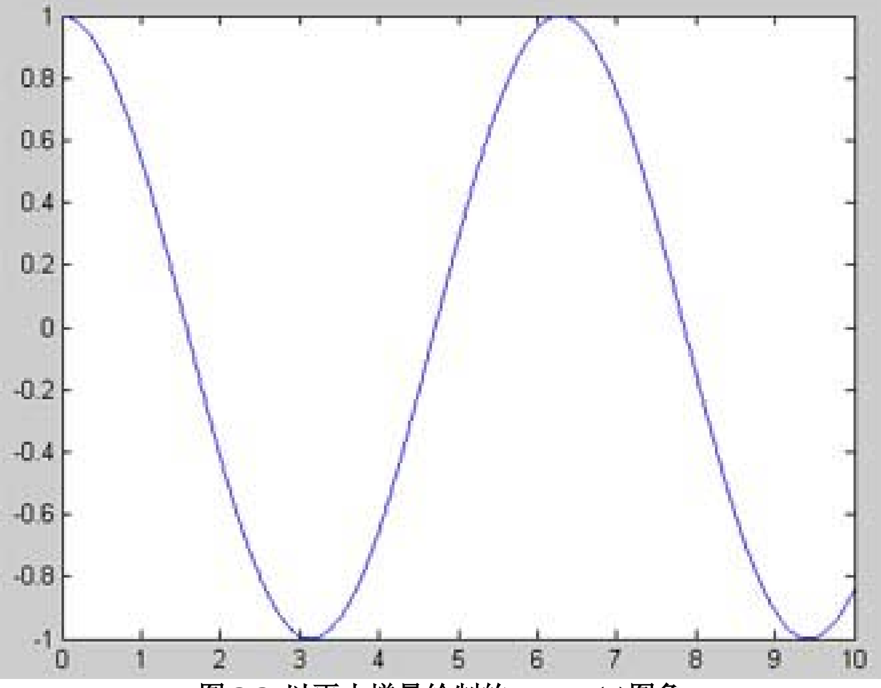


图 3-3 以更小增量绘制的 $y = \cos(x)$ 图象

现在我们知道如何在窗口中直接地绘图了。下一件事你可能就想要绘制一个坐标轴有标签的图象了。这可以通过 `xlabel` 和 `ylabel` 函数做到。这些函数可以带一个用单引号括起来的参数，该参数就是坐标轴的标签。把 `xlabel` 和 `ylabel` 函数用逗号分开与 `plot` 命令放在同一行。例如，下面的命令产生的图象如图 3-4 所示：

```
>> x = [0:0.01:10];
>> y = cos(x);
>> plot(x, y), xlabel('x'), ylabel('cos(x)');
```

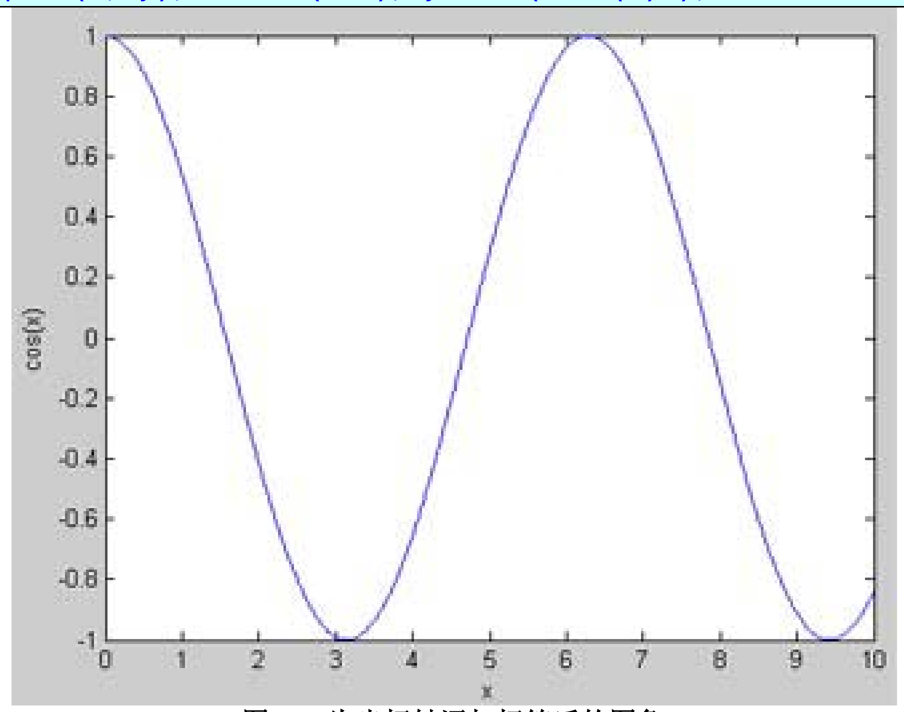


图 3-4 为坐标轴添加标签后的图象

更多 2D 绘图选项

到目前为止，我们知道如何输出函数的一般图象。让我们再看看在绘图时可能会考虑到的选项。如果你要在演示或作业中使用图象，你可能会想要给图象加个标题。**MATLAB** 允许你使用 `title` 命令做到这一点，它使用用单引号括起来的字符串做参数。标题就会在图象的正上方打印出来。假设现在我们需要绘制某个力的图象，这个力遵循 $f(t) = e^{-2t}\sin t$，其中 t 表示时间，单位是秒，范围 $0 \leqslant t \leqslant 4$，每 0.02 秒取一个数据，另外我们还要在图象上显示“阻尼弹力”。怎样做呢？第一步定义时间间隔，以普通的方法做就行了。这里我们用 t 代替 x。

```
>> t = [0:0.02:4];
```

现在我们定义函数，这相当简单：

```
>> f = exp(-2*t)*sin(t);
```

然而，当你这样做的时候，我们得到一条错误消息。**MATLAB** 告诉我们：

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

那么我们如何绕过呢？一种方法是使用 `fplot` 函数来代替，`fplot` 函数会绕过绘图的时间间隔，而自动为我们决定绘图的点数。一般地，`fplot` 为你产生尽可能精确的的图象，同时它也帮助我们绕过像刚才这样的错误。调用 `fplot` 的形式如下

`fplot('function string', [xstart, xend])`

参数 *function string* 告诉 `fplot` 你所要绘制的图象函数，而 *xstart* 和 *xend* 定义了函数的区间。这就简单了，让我们看看如何来求解刚才这个例子。

我们用下面的命令可几步合一，然后敲回车即搞定：

```
>> fplot('exp(-2*t)*sin(t)',[0, 4]);
```

MATLAB 很快就绘制了图象，如图 3-5 所示。

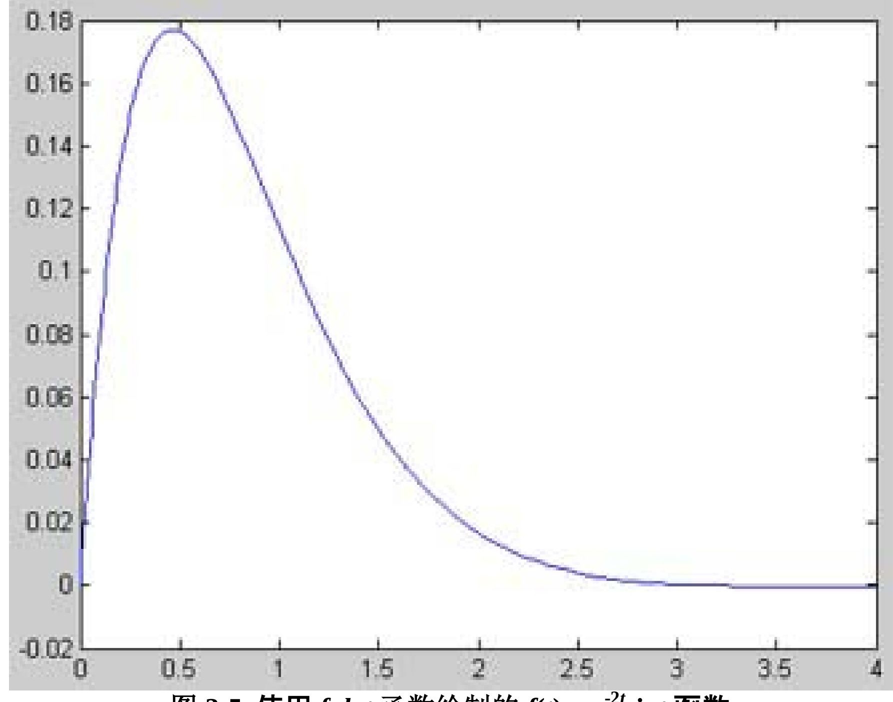


图 3-5 使用 *fplot* 函数绘制的 $f(t) = e^{-2t}\sin t$ 函数



如果我们要为图象添加标签和标题，可以使用与 plot(x, y)相同的后继步骤。我们再做一次，这次添加标题“阻尼弹力”和坐标轴标签。

```
>> fplot('exp(-2*t)*sin(t)',[0, 4]), xlabel('t'),
ylabel('f(t)'), title('阻尼弹力')
```

这行命令产生的图象添加了标签，如图 3-6 所示。

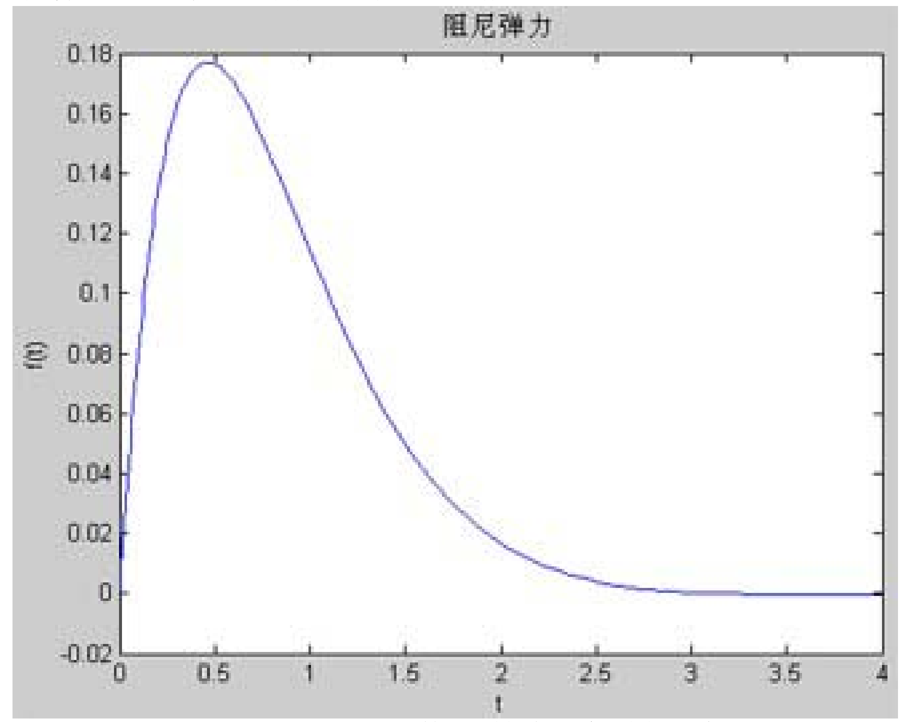


图 3-6 使用 *fplot* 函数绘制的加了标签的图象

刚才我们介绍了 *fplot* 命令，回头再看我们输入指数函数和三角函数相乘时所产生的错误。我们输入：

```
>> t = [0:0.02:4];
>> f = exp(-2*t)*sin(t);
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

在 **MATLAB** 正确的方法是在乘号（*）前带上一个圆句点句（.*），不明白？让我演示一遍正确的输入方法：

```
>> t = [0:0.02:4];
>> f = exp(-2*t).*sin(t);
>> plot(t, f)
```

这一次可就没有什么错误了，也正确地绘制了图象。因此，当一个函数是由二个或更多个函数相乘构成，别忘记在相乘时加上“.”以便告诉 **MATLAB** 我们是对两个矩阵进行相乘。

在本例中，我们用>> f = exp(-2*t).*sin(t);代替>> f = exp(-2*t)*sin(t);，不同的地方是第二部分——我们使用了矩阵相乘（由“.*”指出）。

好了，我们回到老命令 *plot(x, y)*。我们还有没有什么方法可以装扮我们的二维图象呢？一种方法是给图象添加是给图象添加网格，这可以通过在绘图语句中添加 *grid on* 实现。在下一例子中，我们绘出 *y* = tanh(*x*)函数在-6≤x≤6 之间的图象并显示网格。我们先定义间隔：

```
>> x = [-6:0.01:6];
```

接着我们定义函数：

```
>> y = tanh(x);
```

绘图命令看起来如下，而产生的图象如图 3-7.

```
>> plot(x,y), grid on
```

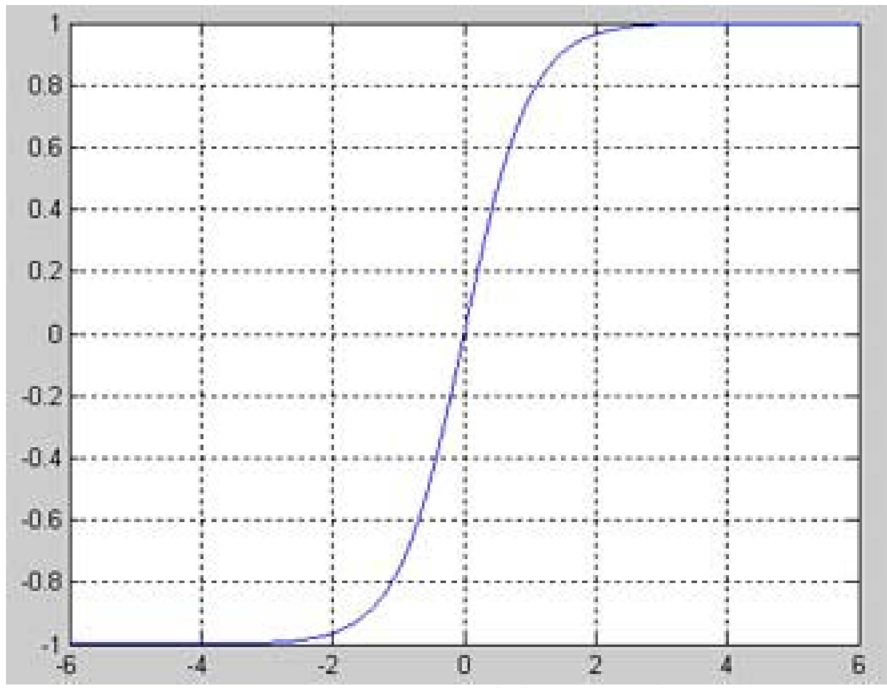


图 3-7 使用了 *grid on* 命令绘制的图象

坐标轴命令

MATLAB 允许你用下面的方式在二维绘图中调整坐标轴。如果我们在绘图命令行中加进 *axis square*，这会使得 **MATLAB** 产生正方形图象。如果我们输入 *axis equal*，那么 **MATLAB** 会产生一个两坐标轴比例和间距都相同的图象。我们回到刚才 *y* = tanh(*x*) 的例子，我们已经把它绘在图 3-7 中。如果你 使用 *axis square* 绘图，你会得到与刚才使用默认设置一样的图象。但假设我们输入：

```
>> plot(x,y), grid on, axis equal
```

在这种情况下，我们得到的图象如图 3-8 所示。注意图 3-7 与图 3-8 中 y 轴的间距有很大不同。在第一种情况中，竖直轴或者说 y 轴的间距与 x 轴的不同。相反，图 3-8 使用的间距是同样的。



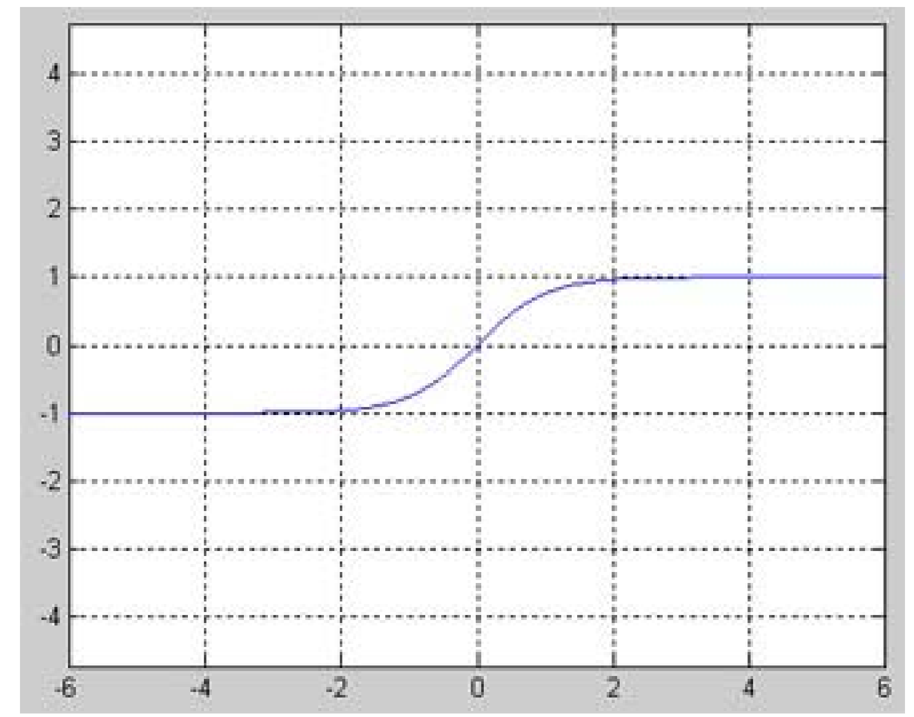


图 3-8 使用了 *axis equal* **选项绘制的** $y = \tanh(x)$**图象**

正如从这个鲜明例子看到的，我们可以使用 `axis` 命令来产生看起来很不一样的图象，因此我们可以使用这个命令产生不同的风格，选用我们所需要的。要让**MATLAB** 自动选择，则输入 `axis auto`。当然，这并不是必须的，除非你已经用过这里所讨论的选项。

在同一图象中显示多个函数

在很多情况下需要在同一个图象中绘制多条曲线。在 **MATLAB** 中要这样做也是相当容易的。我们用在同一个图象中显示两个函数来开始这一节。让我们把 $0 \leqslant t \leqslant 5$ 范围内下面的两个函数绘制在同一个图象中。

$$f(t) = e^{-t}$$
$$g(t) = e^{-2t}$$

我们把 g 函数用虚线绘出以便区别这两条曲线。跟随下面的常规步骤，我们首先定义间隔：

```
>> t = [0:0.01:5];
```

接着我们定义两个函数：

```
>> f = exp(-t);
>> g = exp(-2*t);
```

要绘制多个函数，我们只需调用 `plot(x, y)`函数，其中参数使用一对对的“*x*, *y*”，“*x*, *y*”与“*x*, *y*”之间相互独立，后面跟着用单引号引起来、用来表示我们所要绘制的第二条曲线风格的字符串。在这个例子中我们是：

```
>> plot(t,f,t,g,'--')
```

这告诉 **MATLAB** 要绘制 $f(t)$和 $g(t)$函数，并且第二个函数曲线使用虚线。注意 **MATLAB** 以不同的颜色显示每条曲线。结果如图 3-9 所示。

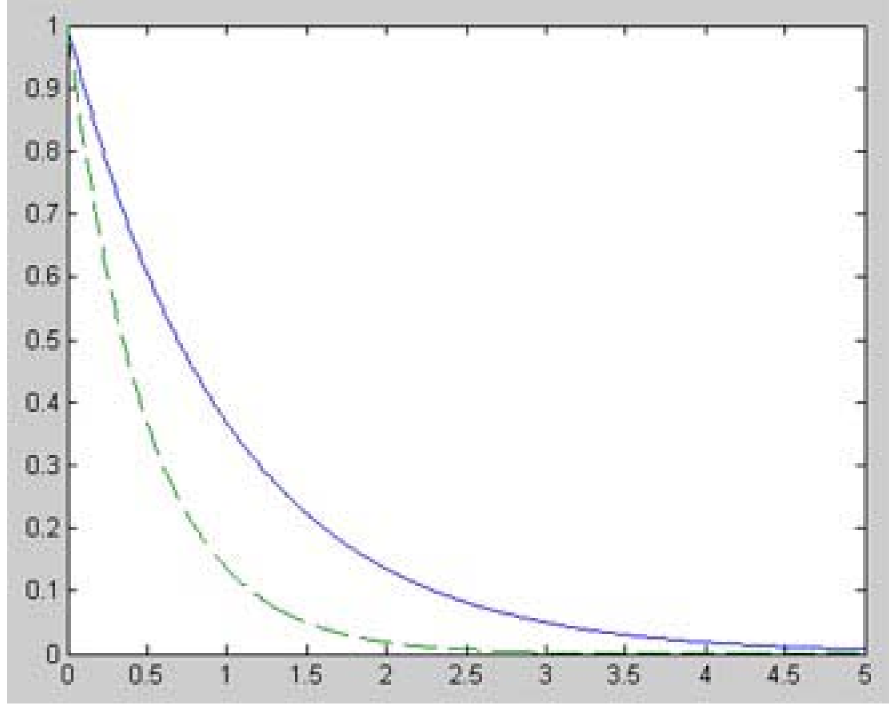


图 3-9 在同一图形中绘制两条曲线

MATLAB 在图象中可以使用四种基本线条风格。它们——放在 *plot* 中用来表示线条风格字符串后面——是：

- 实线 '-'
- 虚线 '--'
- 虚点线 '-.'
- 点线 ':'

让我们用点线表示 f(t) = e^{-t} 函数，重新绘制图 3-9。命令是

```
>> plot(t,f,':',t,g,'--')
```

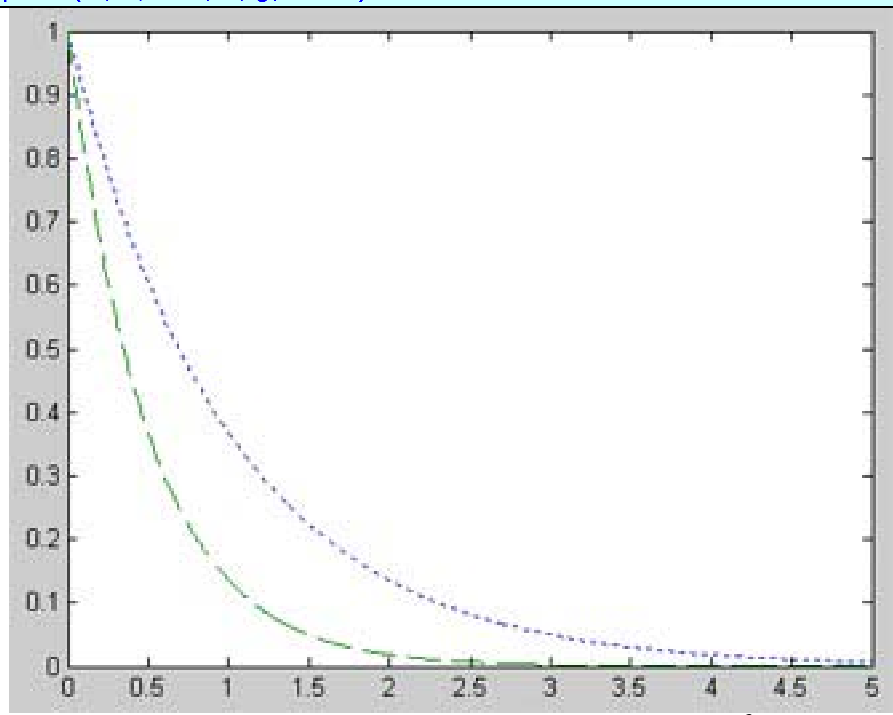


图 3-10 用点线表示 $f(t) = e^{-t}$，用虚线表示 $g(t) = e^{-2t}$

产生的图象如图 3-10 所示。

如果你想让曲线全部用实线表示而只是让颜色不同而已，那么就把表示曲线类型的字符

串省略掉。图象就会使用实线绘制——这是默认设置。

添加图例

专业的图象总是附有图例，告诉读者某个曲线是什么。在下面的例子中，假设我们要绘制两个表示势能的函数，它们由双曲三角函数 *sinh*(*x*)和 *cosh*(*x*)定义，定义域为 0≤*x*≤2。首先我们定义 *x*：

```
>> x = [0:0.01:2];
```

现在我们定义这两个函数，在 **MATLAB** 中把函数称为 *y* 并不是什么不可思议的事，所以我们把第二个函数称为 *z*，因此有：

```
>> y = sinh(x);
>> z = cosh(x);
```

legend 命令用起来很简单。只需把它加在 *plot*(*x*,*y*)命令后面，并用单引号把你要添加为图例的文本引起来。在这个例子中我们有：

legend('sinh(x)','cosh(x)')

我们只需把这一行添加到 *plot* 命令后面。在这个例子中，我们还包含 *x* 和 *y* 标签，第一条曲线用实线而第二条曲线用虚点线：

```
>> plot(x,y,x,z,'-.'), xlabel('x'), ylabel('Potential'),
>> legend('sinh(x)','cosh(x)')
```

结果如图 3-11 所示。把图例移到一个合适位置对于打印或看起来可能更好，此时只需用鼠标图例拖到你想要的地方即可。

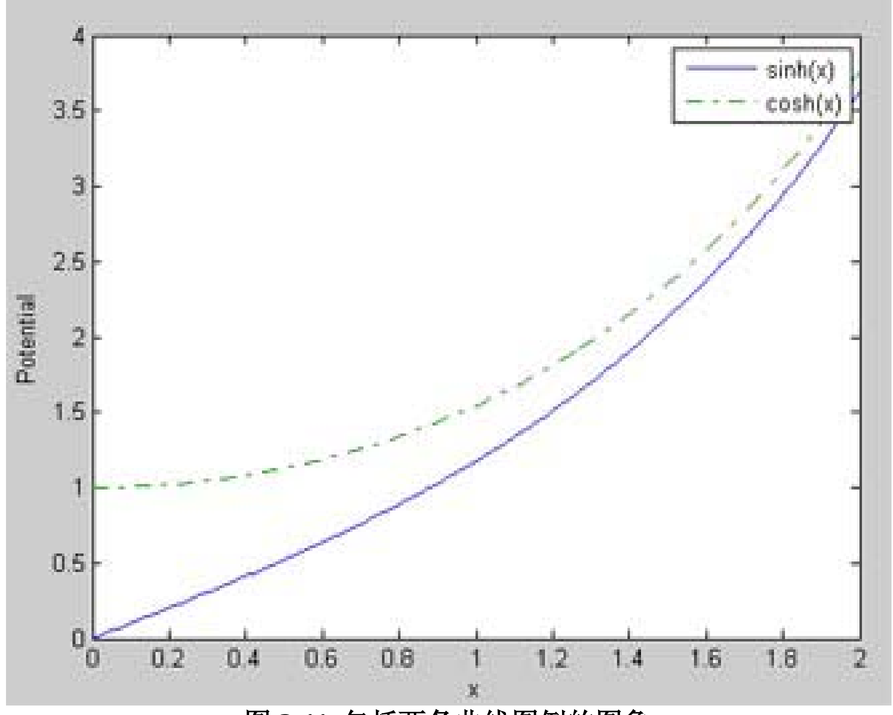


图 3-11 包括两条曲线图例的图象

设置颜色

每条曲线的颜色 **MATLAB** 可以自动设置，我们也选择自己的颜色。通过在 *plot* 命令中指定 **MATLAB** 所使用的表示颜色的字符即可。让我们用一个例子来演示。

我们再次绘制双曲正弦和余弦函数。这次我们为图象使用不同的区间，选 $-5 \leq x \leq 5$。所以定义我们的数组为：

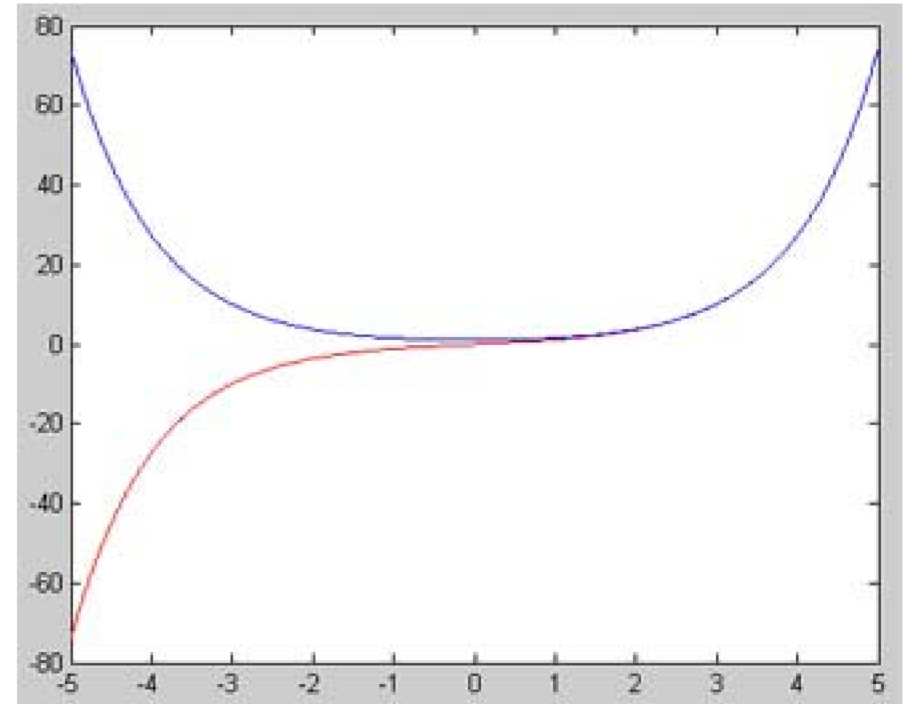
```
>> x = [-5:0.01:5];
```

现在我们重新定义函数。记得如果我们不做这一步，我们仍然处在 **MATLAB** 的同一会话中。程序会认为函数仍然采用先前定义的 *x* 区间中。所以我们输入：

```
>> y = sinh(x);
>> z = cosh(x);
```

现在我们用红曲线表示 *y* 蓝曲线表示 *z* 重新绘图。我们通过在 *plot* 函数中 *y* 和 *z* 入口后面分别用字符 *r* 和 *b* 表示颜色。命令如下：

```
>> plot(x,y,'r',x,z,'b')
```



在你自己的系统上试试，看看它所产生的图象。现在要为每条曲线设置多于一个选项变得可能了。让我们把曲线设置为红色和蓝色，并把 cosh 函数（蓝色曲线）设为虚线。用下面的方法设置即可——把为某条曲线设置的所有绘线选项都用单引号引起来：

```
>> plot(x,y,'r',x,z,'b--')
```

这个命令给我们画出的图象如图 3-12 所示。



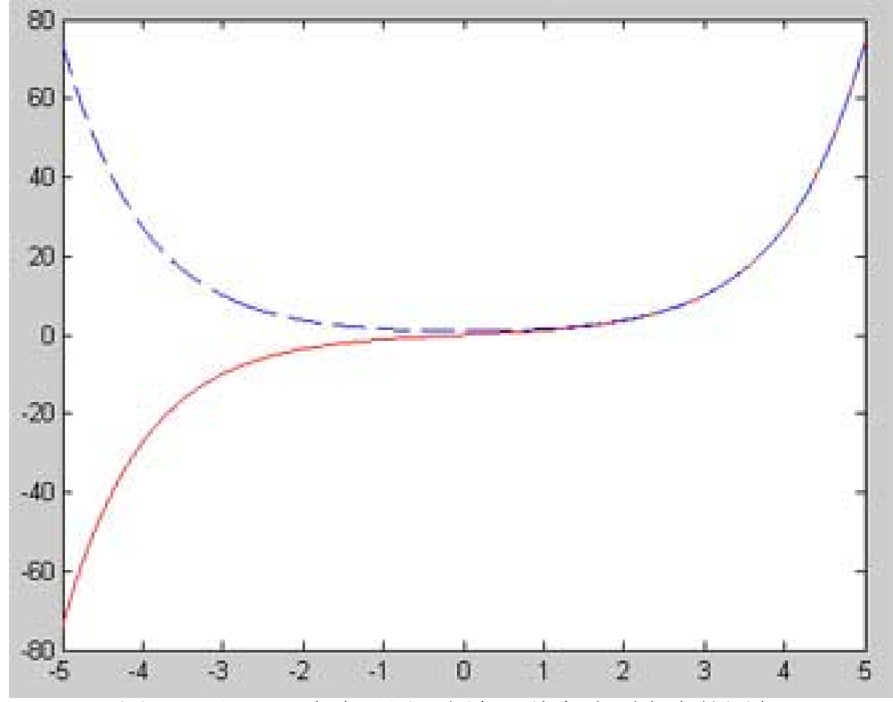


图 3-12 用同一命令设置了颜色和线条类型产生的图象

MATLAB 给用户绘制图象八种颜色选择。它们如表 3-1 所示。

表 3-1 **MATLAB** 颜色说明符

颜色	说明符
白色	w
黑色	k
蓝色	b
红色	r
青色	c
绿色	g
洋红	m
蓝色	y

设置坐标比例

另外我们看一下 *axis* 命令如何设置绘图范围。可以用下面的方式调用 *axis* 命令：

axis ([xmin xmax ymin ymax])

假设我们要产生函数 $y = \sin(2x + 3)$在 $0 \leq x \leq 5$ 之间的图象。我们可能会考虑函数的 y 值在-1 到 1 之间，那么就可以设置 y 轴仅显示这些值。使用下面格式的命令：

```
>> x = [0:0.01:5];
>> y = sin(2*x + 3);
>> plot(x,y), axis([0 5 -1 1])
```

这些命令产生的图象如图 3-13 所示。

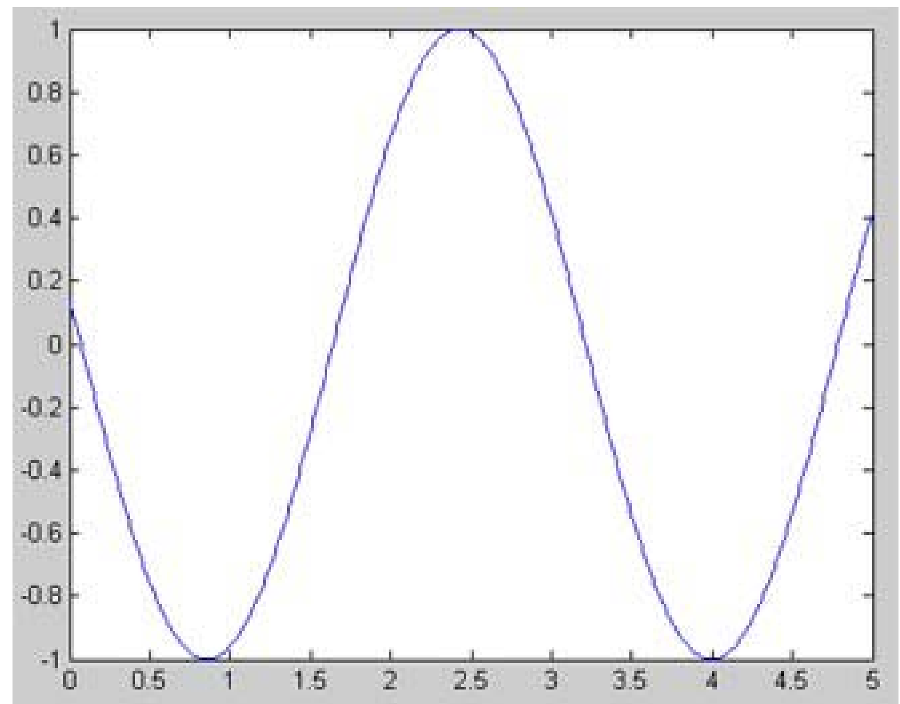


图 3-13 函数 $y = \sin(2x + 3)$在 $0 \leqslant x \leqslant 5$ 范围内被设置了 y 轴和 x 轴范围的图象

现在我们绘制 $y = e^{-3/2x}\sin(5x + 3)$的图象。先试一下 $0 \leqslant x \leqslant 5$ 和 $-1 \leqslant y \leqslant 1$

```
>> y = exp(-1.5*x).*sin(5*x + 3);
>> plot(x,y), axis([0 5 -1 1])
```


产生的图象如图 3-14 所示。正如你从图形所看到的，y 轴的范围可以再调整。

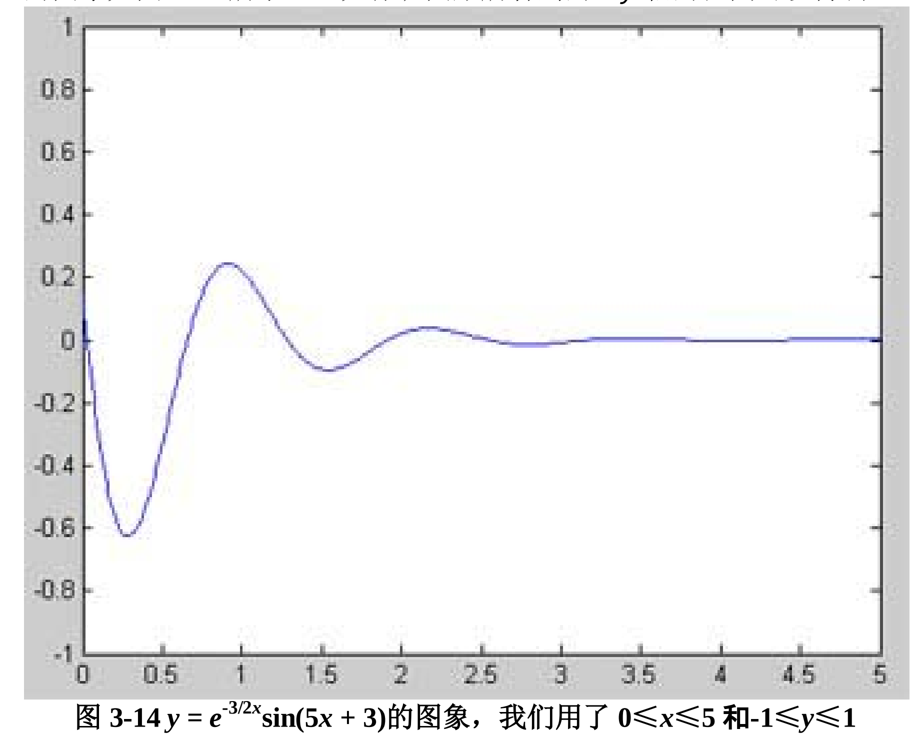


图 3-14 $y = e^{-3/2x}\sin(5x + 3)$的图象，我们用了 $0 \leqslant x \leqslant 5$ 和$-1 \leqslant y \leqslant 1$

我们尝试调整一下图象 y 值的范围，取$-0.7 \leqslant y \leqslant 0.3$。把 *axis* 命令变成下面的形式：

```
>> plot(x,y), axis([0 5 -0.7 0.3])
```



现在图形看起来更紧凑了，如图 3-15。

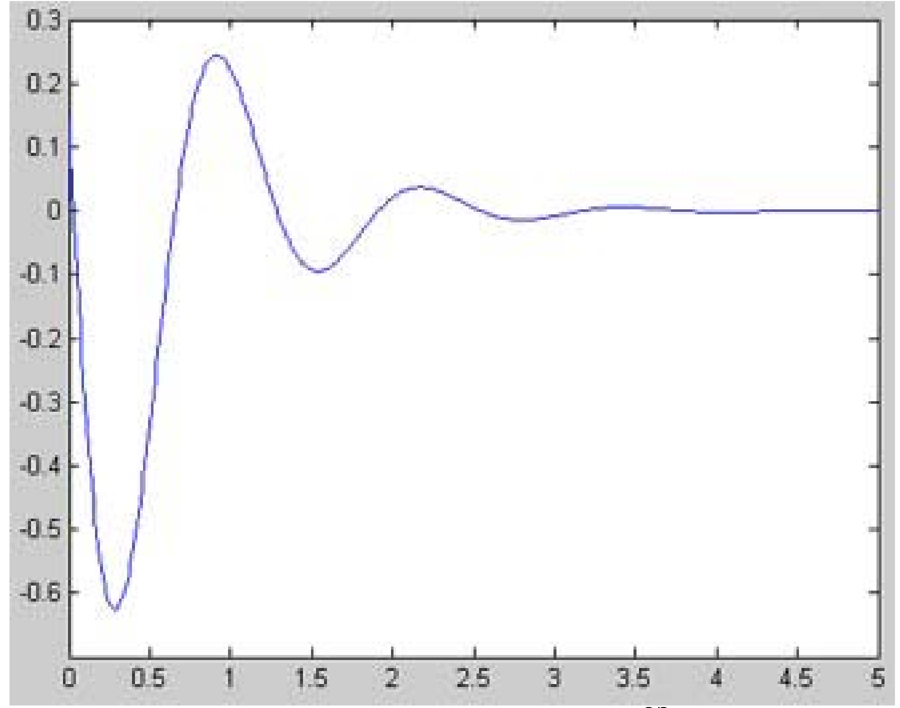


图 3-15 用 *axis*([0 5 -07 0.3])设置 *y* 区间后的 $y = e^{-3/2x}\sin(5x + 3)$图象

下面我们绘制 $y = \sin^2(5x)$的图象。做为插入语，在 **MATLAB** 中我们如何对 sin 函数进行平方呢？如果我们输入：

```
>> y = sin(5*x)^2
```

MATLAB 将甩出

```
??? Error using ==> mpower
Matrix must be square.
```

对 sin 函数平方的正确方法是使用数组相乘记号，即使用 ***A*.^*B*** 表示 A^B。因此下面的命令才能正确工作。

```
>> y = sin(5*x).^2;
```

它对数组中的每个元素进行平方而不是对整个数组进行平方。现在我们使用默认设置来绘图。**MATLAB** 产生的图象如图 3-16 所示。

假设我们想要看看 *x* 在某个区间内的图象。例如，我们把 *x* 设为 $0 \leqslant x \leqslant 1$。输入如下：

```
>> plot(x,y), axis([0 1 0 1])
```

产生的图象如图 3-17 所示。

到此，你可以在**MATLAB**处理基本的绘图了。接下来我们学习在同一个图中放置两个或多个图象的情况。

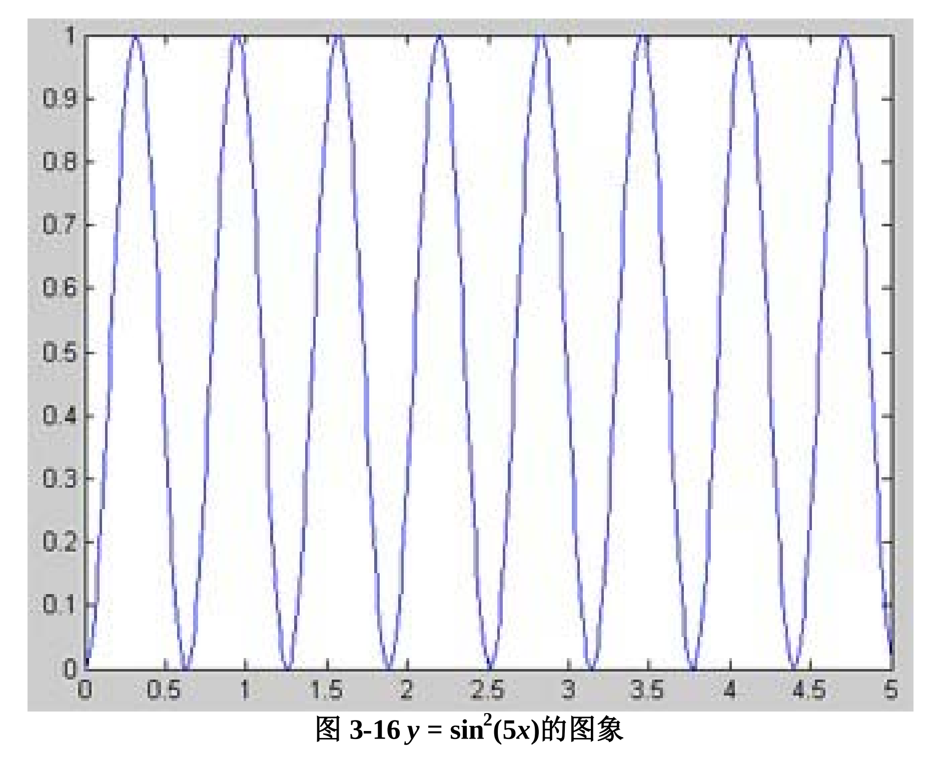


图 3-16 $y = \sin^2(5x)$的图象

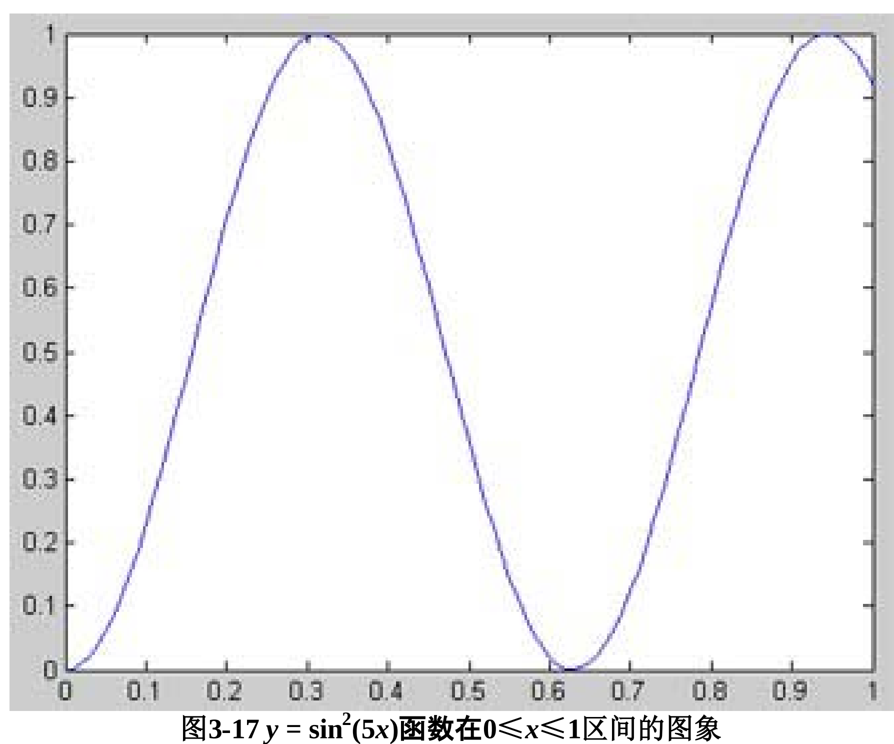


图3-17 $y = \sin^2(5x)$函数在$0 \leqslant x \leqslant 1$区间的图象

子图

子图

子图即是要在一个图上显示多于一个图象。绘制子图使用命令 `subplot(m, n, p)`，这里 *m* 和 *n* 告诉 **MATLAB** 产生的产生的子图有 *m* 行和 *n* 列，*p* 用来告诉 **MATLAB** 我们所要贴上去的某个已经绘制的图形窗口。与以往一样，我们最好用例子演示说明。

用 *subplot* 命令创建的图象都有它自己的特性。第一个例子我们并排显示 *y* =



$e^{-1.2x}$sin(20x)和 y = e^{-2x}sin(20x)的图象。在这两种情况下，我们都设为 0≤x≤5 和 -1≤y≤1。首先我们定义函数的定义域、函数然后再调用 *subplot*：

```
>> x = [0:0.01:5];
>> y = exp(-1.2*x).*sin(20*x);
>> subplot(1,2,1)
```

通过给 *subplot* 传递(1, 2, 1)，我们告诉 **MATLAB** 我们要创建的图有 1 行 2 个窗格块（1 行 2 列）。接下来特定的图象将显示在第一个窗格块中。这些窗格块以常规的方式的从左到右编以数字，因此这个图象将显示在左边的窗格块中，不过还没有在它上面放置任何东西。如图 3-18。

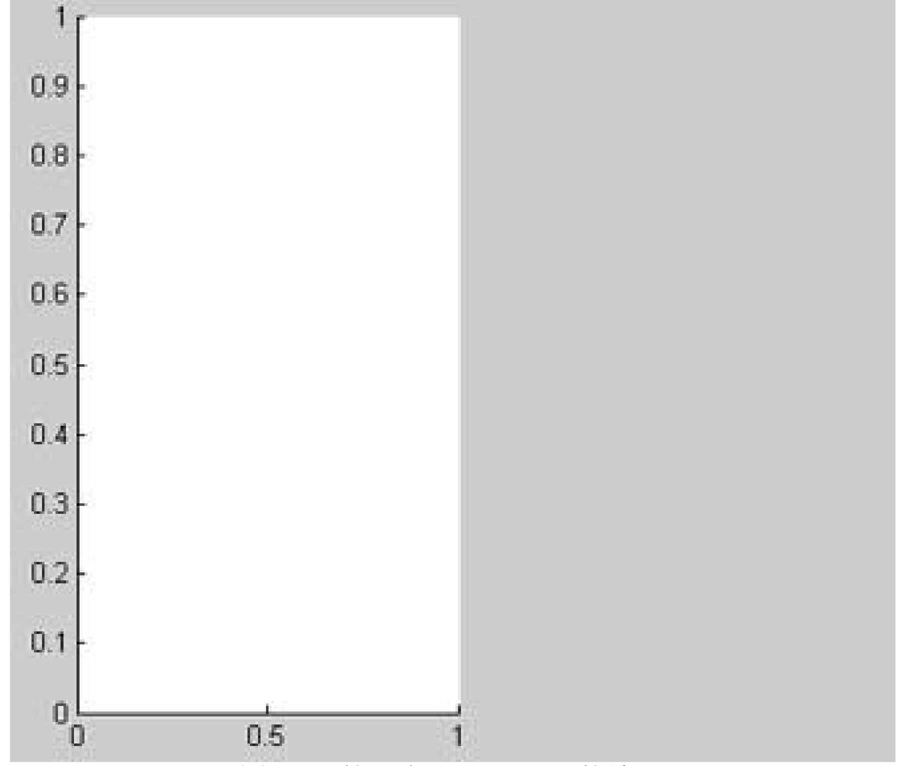


图 3-18 第一次调用 *subplot* 的结果

现在我们调用 *plot* 命令：

```
>> plot(x,y),xlabel('x'),ylabel('exp(-1.2x)*sin(20x)'),
axis([0 5 -1 1])
```

如果现在看图，函数的图象已经在第一个窗格块中被绘制了。如图 3-19 所示。

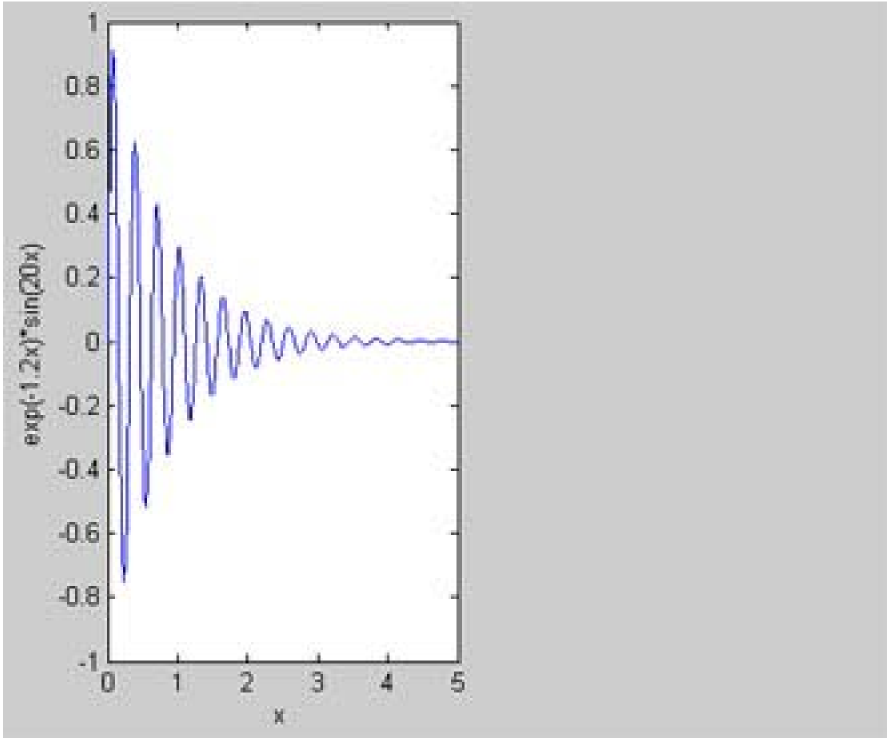


图 3-19 我们调用 *subplot* 和 *plot* 命令后的情况

随着第一个图象创建完成，我们可以继续创建第二个图象。首先定义函数：

```
>> y = exp(-2*x).*sin(20*x);
```

子图

（插入语——记得两个函数相乘时在"*"前面加上点句号"."）。现在我们调用 `subplot`，这一次告诉 **MATLAB** 把第二个函数的图象放在第二个窗格块中：

```
>> subplot(1,2,2)
```

MATLAB 还没有放上任何东西——记住，我们还必须调用 `plot` 命令。现在图看起来像图 3-20。

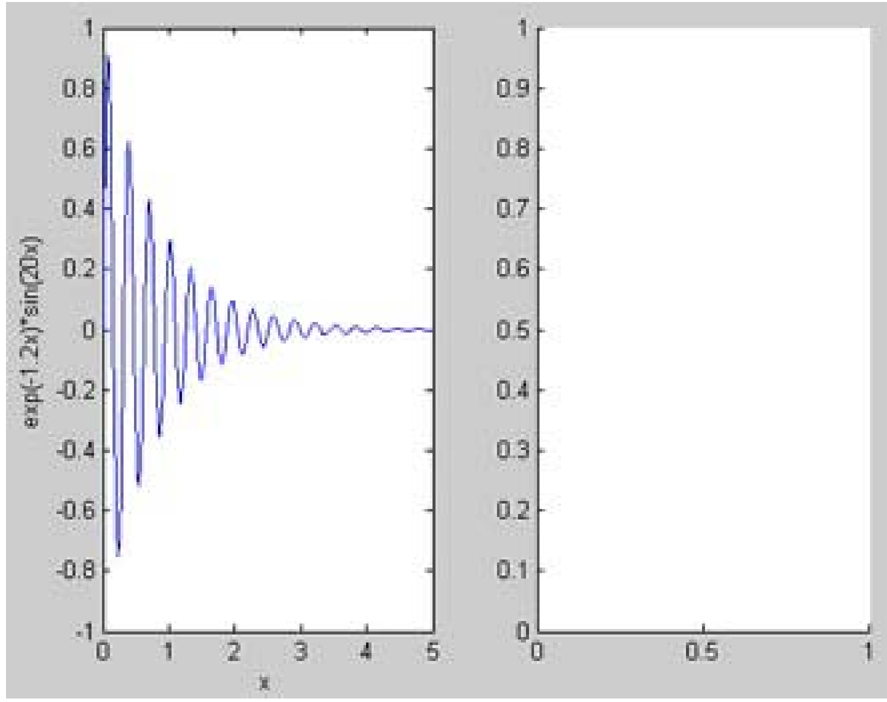


图 3-20 我们第二次调用 subplot 后的图形窗口

我们绘制它：

```
>> plot(x,y),xlabel('x'),ylabel('exp(–2x)*sin(20x)'),
axis([0 5 -1 1])
```

结果，两个并排的图象如图 3-21 所示。

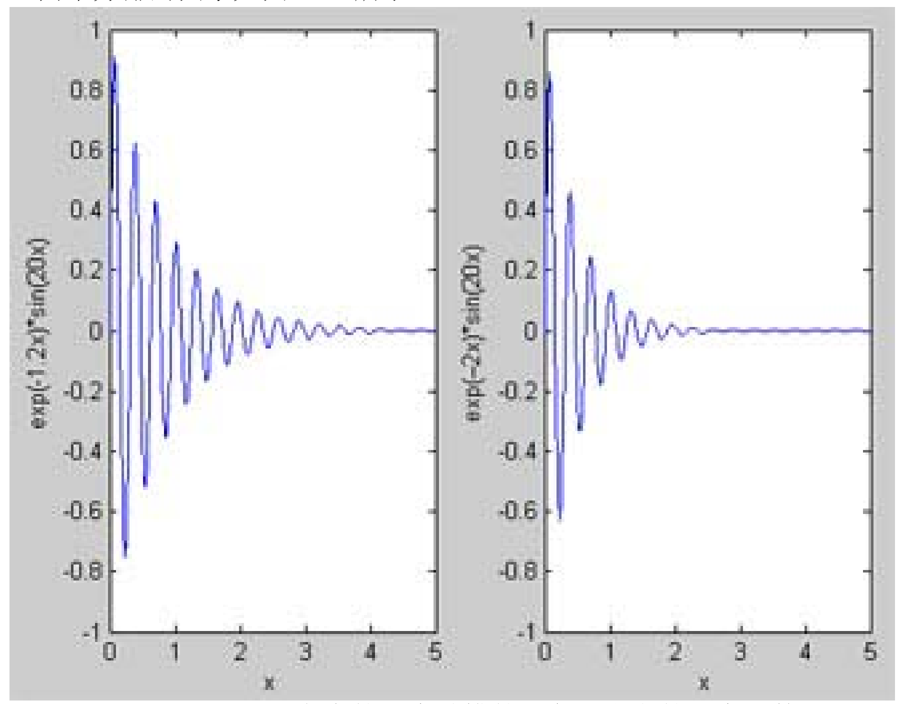


图 3-21 MATLAB 产生的两个并排的图象。定义第一个函数，调用 subplot，告诉 MATLAB 哪里放置它，然后调用 plot 绘制画布。然后重复第二个函数。

图象重叠和 *linspace* 命令

假设我们绘制了一个函数的图象，然后又决定在同一个图形上再绘制另一个函数的图象。我们通过告诉 **MATLAB** `hold on` 后两次调用 `plot` 命令即可做到。

在下面的例子中，我们将绘制 cos(*x*)和 sin(*x*)的图象再把它们放在同一图形上。首先，我们学习一个用来产生 *x* 数集的新命令，即 `linspace` 命令，它可以以两种方式调用。如果我们写成：

```
x = linspace(a,b)
```

MATLAB 会在 *a* 到 *b* 间取出均匀分布的 100 个点（行向量），如果写成

```
x = linspace(a,b,n)
```

那么 **MATLAB** 会在 *a*、*b* 之间取出均匀分布的 *n* 个点。现在我们使用这个工具来绘制 cos(*x*)和 sin(*x*)的图象。我们用下面的命令在 0 到 2π 之间间隔均匀地取出 100 个点：

```
>> x = linspace(0,2*pi);
```

现在我们绘制 cos(*x*)：

```
>> plot(x,cos(x))
```

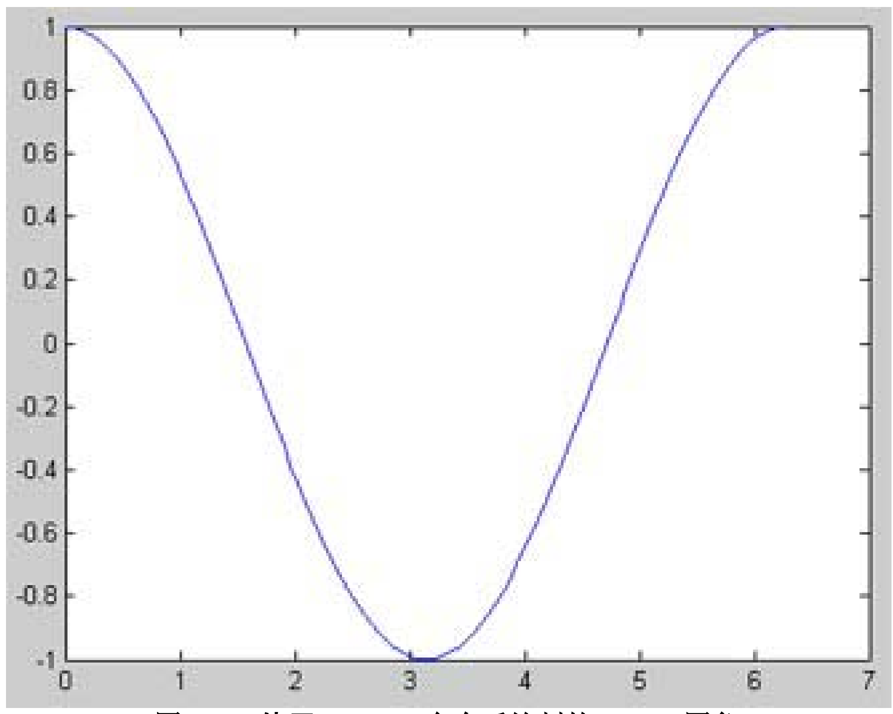


图 3-22 使用 *linspace* 命令后绘制的 cos(*x*) 图象

我们产生的图象如图 3-22 所示。如果现在输入：

```
>> plot(x,sin(x))
```

MATLAB 会把先前的输出冲掉。现在图形窗口变成图 3-23。

快速便道——注意虽然我们定义了 *x* 的范围在 $0 \leqslant x \leqslant 2\pi$ 之间，**MATLAB** 计算的图形比函数所计算图象宽了一些。我们可以在调用 `plot(x,sin(x))`时用 `axis` 命令进行修正：

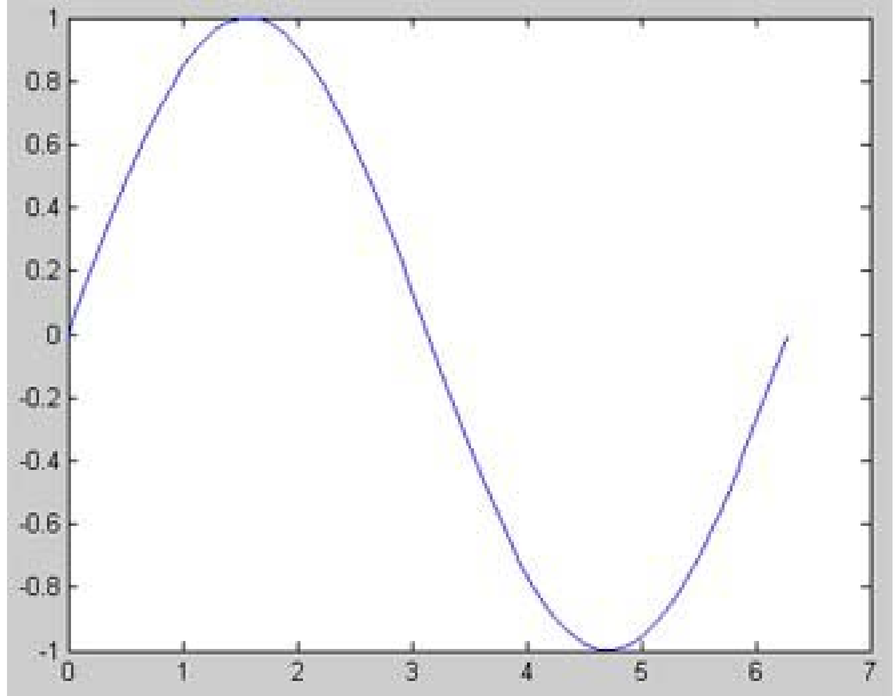


图 3-23 我们输入 *plot*[*x*, sin(*x*)]命令后冲掉了先前绘制的图象

```
>> plot(x,sin(x)),axis([0 2*pi -1 1])
```

现在产生的图象好看一些了，如图 3-24。



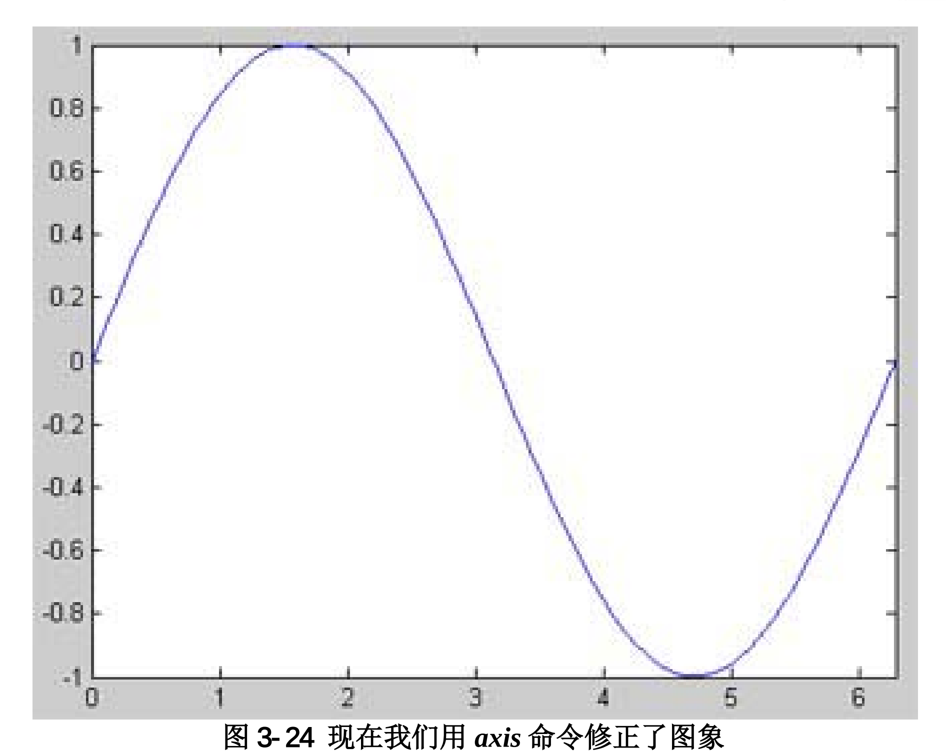


图 3-24 现在我们用 *axis* 命令修正了图象

回到我们的困境，我们要在一个图形上绘制 cos(*x*)的同时再绘上 sin(*x*)。可以通过下面的命令组做到：

```
>> x = linspace(0,2*pi);
>> plot(x, cos(x)),axis([0 2*pi -1 1])
>> hold on
>> plot(x, sin(x)), axis ([0 2*pi -1 1])
```

结果如图 3-25 所示，在同一个图形上显示两条曲线。如果需要，你可以用我们前面所介绍的选项为每曲线选择线条类型和颜色。

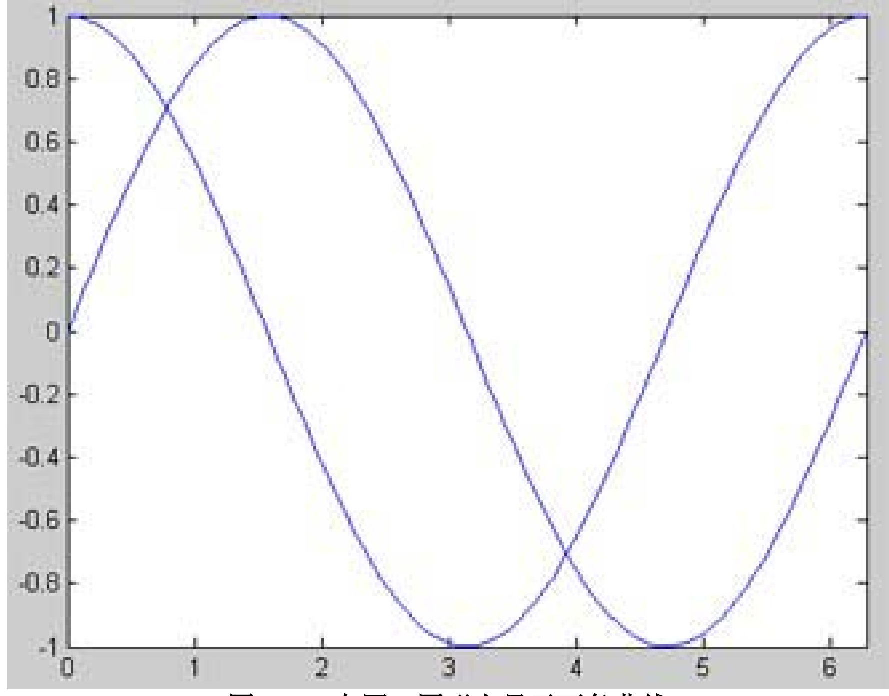


图 3-25 在同一图形上显示两条曲线

极坐标和对数图象

如果你使用过微积分，毫无疑问你不会对极坐标图象和对数图象感到陌生。回想我们用手工绘制这些图象的时候——如果可以用计算机程序进行绘制或检验你的答案岂不大快人心？谢天谢地，**MATLAB** 可以做到。我们先看看极坐标图象，它绘制半径 r 和角度 θ 之间的图象。

作为第一个例子，我们绘制一条螺线——阿基米德螺线——它由下面的简单关系构成：

$$r = a\theta$$

其中 a 是一个常数。我们绘制 $a = 2$ 和 $0 \leq \theta \leq 2\pi$ 的极坐标图象。第一个语句，我们定义常数：

```
>> a = 2;
```

这确实是够简单的了。接着我们定义函数 $r(\theta)$，这需要两步完成，第一步把 θ 与我们前面例子中的独立变量 x 一样看待，所以我们要定义它的名称、区间和所要使用的增量，第二步我们再定义 r：

```
>> theta = [0:pi/90:2*pi];
>> r = a*theta;
```

这些语句告诉 **MATLAB** `theta` 定义在 $0 \leq \theta \leq 2\pi$ 之间，并选择增量为 $\pi/90$。产生极坐标图象的命令为：

`polar ( theta, r)`

现在我们调用它，同时给它添加一个标题：

```
>> polar(theta,r), title('阿基米德螺线')
```

结果如图 3-26 所示。

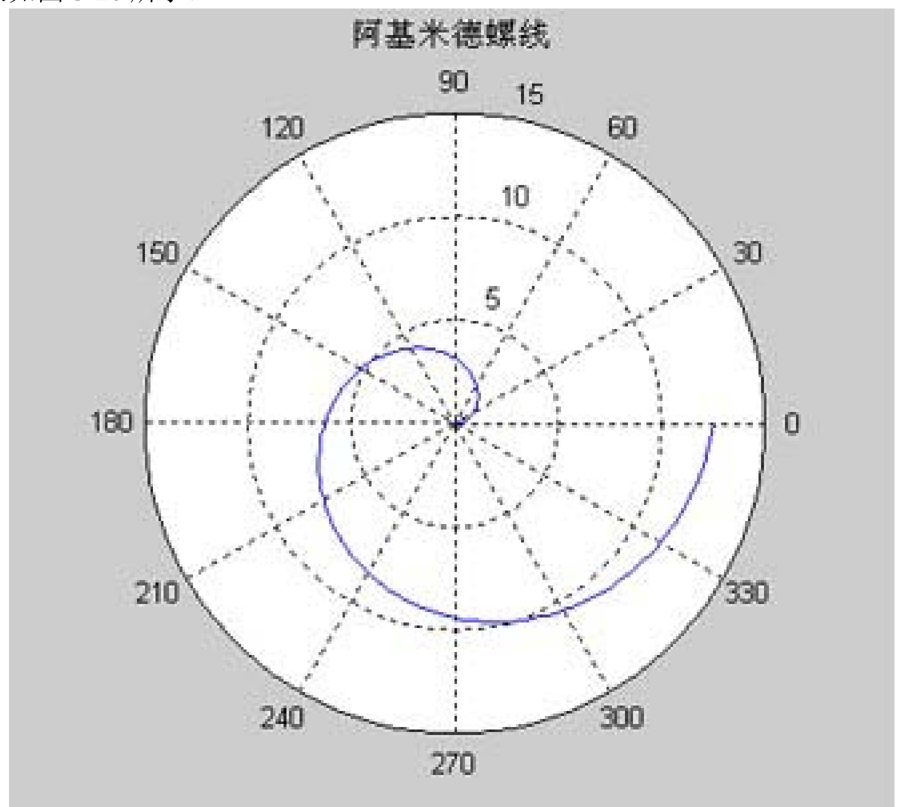


图 3-26 极坐标系中的阿基米德螺线图象

很多 `plot` 可用的选项 `polar` 同样可用。



第二个例子，假设我们要绘制下面函数的极坐标图象：

$$r = 1 + 2\cos\theta$$

这里 $0 \leqslant \theta \leqslant 6\pi$，并用虚线绘制。首先我们定义 θ 区间：

```
>> theta = [0:pi/90:6*pi];
```

现在输入函数 $r(\theta)$：

```
>> r = 1 + 2*cos(theta);
```

现在告诉 **MATLAB** 用练色虚点线绘制曲线。如下输入：

```
>> polar(theta,r,'r-.')
```

产生的图象如图 3-27 所示。

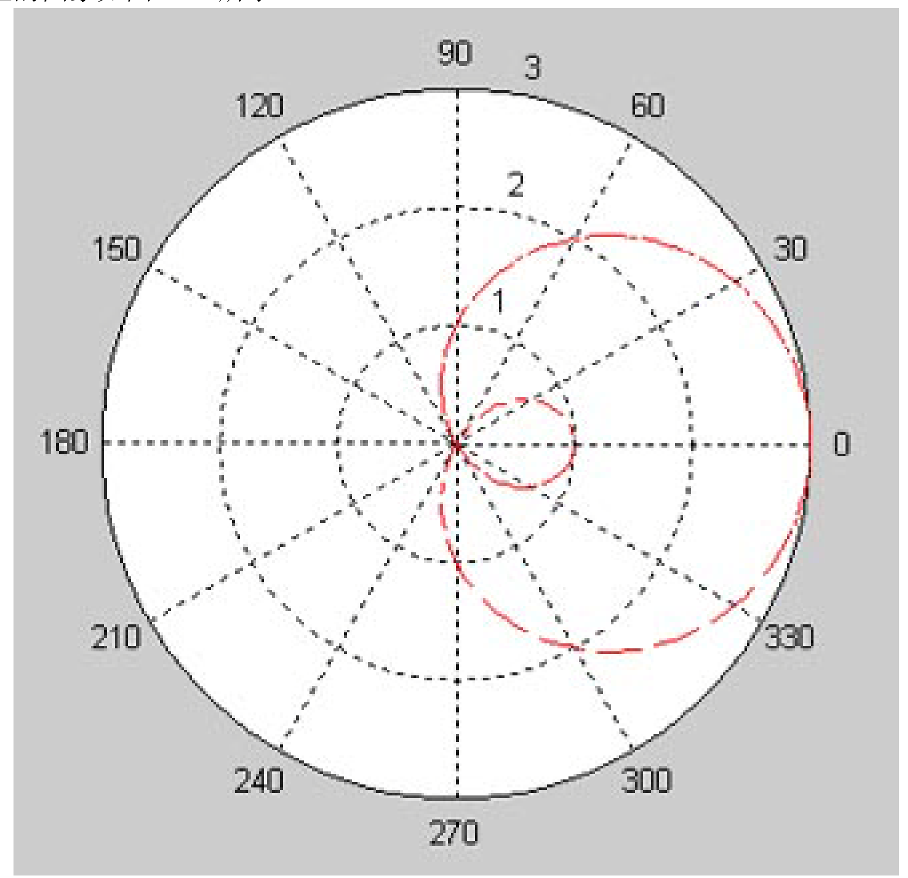


图 3-27 $r = 1 + 2\cos\theta$ 函数的极坐标图象

现在我们来看看如何用 **MATLAB** 来绘制对数图象。这曾经让我头痛。如果你是电子工程师，你会发现这个特性很有用。第一种我们能使用的对数图象是 `log-log` 图象。我们用电子中的一个典型电路例子看看如何使用它。这个电路含有一个电压源、电阻和电容。由于很多读者未必是电子工程师，所以我们不讨论这个方程是如何得到的。在这里我们的目的只是看看如何产生 `log-log` 图象。

事实表明，在一个电路中如图输入电压是正弦信号并遵循 $v_i = A_i\sin\omega t$，那么输出电压将会是另一个正弦函数，可写成 $v_o = A_o\sin(\omega t+\varphi)$。其中电子工程师可能会关注的地方是电路的**频率响应**，即是输出对输入的比率（放大倍数），它们存在下面的关系：

$$\frac{A_o}{A_i} = \left|\frac{1}{1 + i\omega RC}\right|$$

一般地，频率响应告诉我们在不同的频率下输出信号对输入信号的增强程度。由于电子工程师很喜欢用拉普拉斯变换，因此通常让 $s = i\omega$。我们取 ω 的范围为 $1 \leqslant \omega \leqslant 100$ *rad/s*，

电阻 R 和电容 C 的乘积 RC 的单位为秒。对于我们的例子，让 $RC = 0.25$ 秒。下面在 **MATLAB** 定义这些量：

```
>> RC = 0.25;
>> s = [1:100]*i;
```

注意在第二行中，我们把 s 定义为复数变量。频率响应是一个**输出/输入**与**频率**间的对数关系，所以让我们定义 A_o/A_i，注意在定义式中我们要用的是绝对值，在 **MATLAB** 中我们要把函数传递给 abs 命令：

```
>> F = abs(1./(1+RC*s));
```

现在产生图象的所有条件都有了。同样的，plot 的很多选项在 loglog 中也可以使用。你可能想在对数图象中显示网格，还可以包含坐标标签和图象标题：

```
>> loglog(imag(s),F),grid,xlabel('频率 (rad/s)'),
ylabel('输出/输入比'),title('频率响应')
```

这行命令产生了非常漂亮的对数图象，如图 3-28 所示。

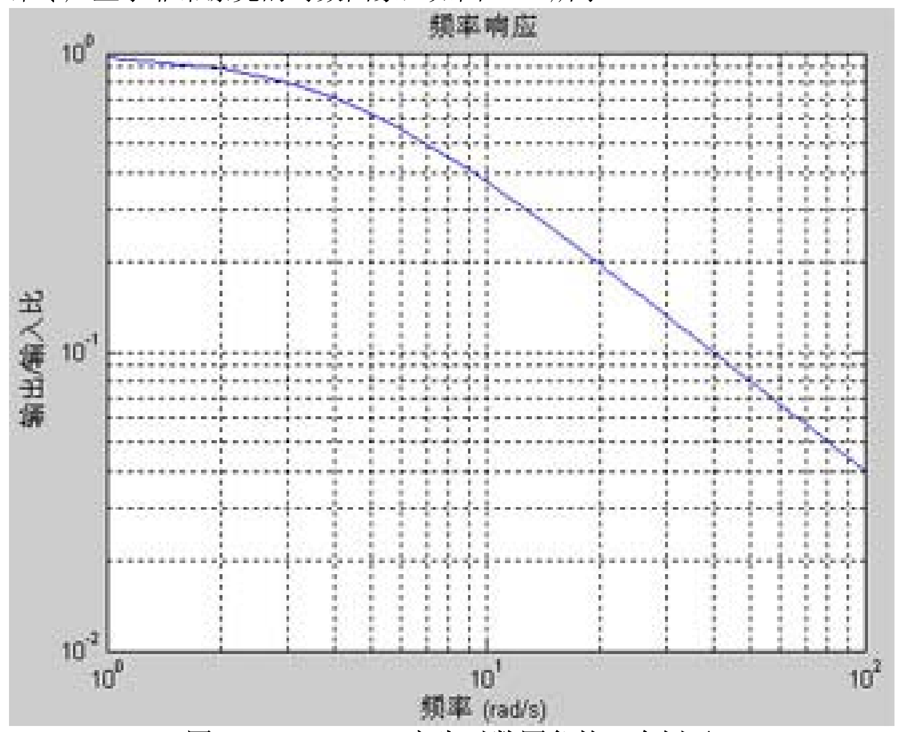


图 3-28 MATLAB 产生对数图象的一个例子

依我看，log-log 图象非常容易绘制，输出结果也非常漂亮。使用 log-log 图象的一种情况是当给定的函数在定义域的一个小范围内变化非常快。让我们看另外一个简单例子。

我们考虑 $y = e^{-10x^2}$ ，其中 $0 \leq x \leq 20$。我们可以用常用的方法绘制它，命令如下：

```
>> x = [0:0.1:20];
>> y = exp(-10*x.^2);
>> plot(x,y);
```

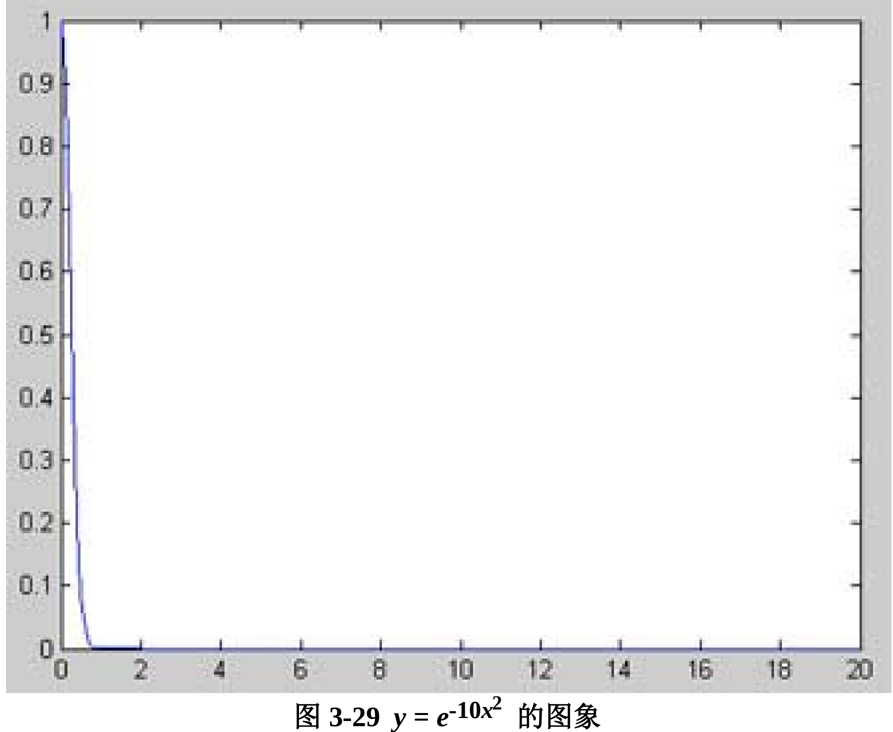


图 3-29 $y = e^{-10x^2}$ **的图象**

从图 3-29 可以看到，在数据集很小的一个范围之内发生了所有的情况。让我们试一下对数图象。输入：

```
>> loglog(x,y)
```

这命令产生的图象如图 3-30。注意函数的特性在整个数据范围内已经被完全表现出来（无穷小部分除外）。

我们还有另外两个选择，第一个是 *semilogx(x, y)*，它产生的图象 *x* 轴使用对数值，*y* 轴仍然用直接值；相应地，*semilogy(x, y)*产生的图象 *y* 轴使用对数值，*x* 使用直接值。

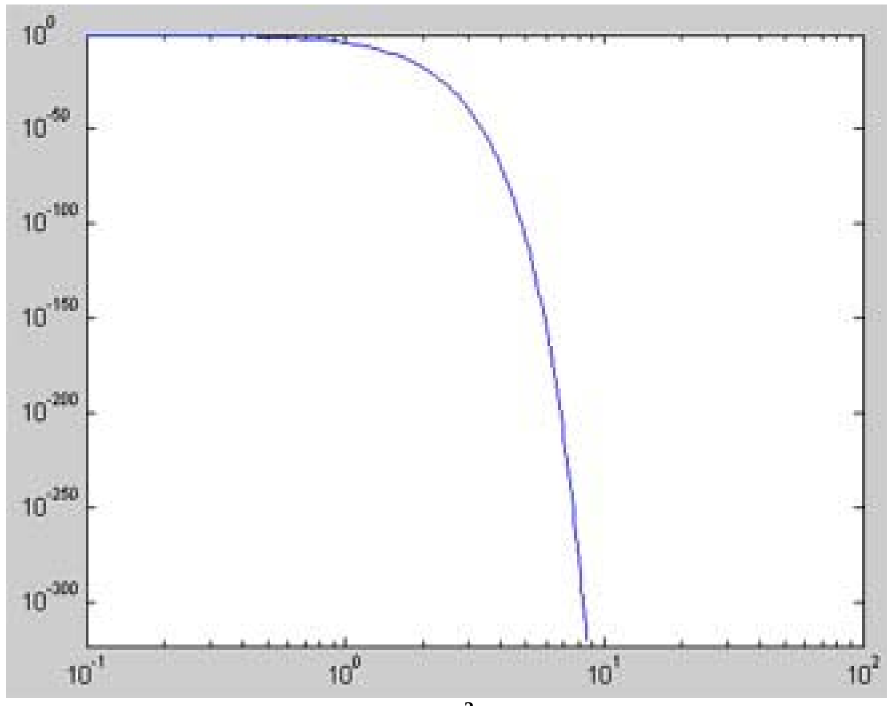


图 3-30 $y = e^{-10x^2}$ **的对象图象**

离散数据绘图

现在让我们开始接触 **MATLAB** 中一些在你的工作领域中可能会用到的特殊应用。首先我看看如何使用 *plot*(*x*, *y*)绘制一些离散数据图并用线把它们连接起来。假设某个班有 5 位同学：Adrian、Jim、Joe、Sally 和 Sue，他们在考试中得到的分数分别为 50、98、75、80 和 98。我们如何绘制这些数据的图象呢？

第一件事是定义两个数组，包含了学生名单和考试分数。学生名单在本例中充当 *x*，因此我们就可以创建含有 5 个元素的 *x* 数组。

```
>> x = [1:5];
```

由于我们不是对一个连续函数进行建模，因此没有必要指定增量。我们使用的增量默认为 1，因此 **MATLAB** 继续运行，为我们产生了 5 个点。现在我们为每个点填上相应的分数，即是 *y* 值——我们只需把它们写成为一个行向量。输入时方括号内使用逗号把各测试分数隔开：

```
>> y = [50,98,75,80,98];
```

现在我们可以告知 **MATLAB** 所要使用的标签，进行数据绘图。使用下面的命令集，看起来有些复杂：

```
>> plot(x,y,'o',x,y),set(gca,'XTicklabel',['Adrian'; 'Jim';'Joe';'Sally';'Sue']),
set(gca,'XTick',[1:5]), axis([1 5 0 100]),xlabel('学生'), ylabel('期末成绩'),title('2005 年 12 月期末考试')
```

很不幸，这样做会得到下面的错误：

```
??? Error using ==> vertcat
All rows in the bracketed expression must have the same number of columns.
```

这表明我们不能使用学生姓名，因为列表中的姓名元素必须具有相同的字符数。这里我们为每个学生分配一个 ID，即使用下面的语句赋值：

['Adrian'; 'Jim';'Joe';'Sally';'Sue'] = ['001';'002';'003';'004';'005']

现在命令看起来：

```
>> plot(x,y,'o',x,y),set(gca,'XTicklabel',['001'; '002';'003';'004';'005']),
set(gca,'XTick',[1:5]),axis([1 5 0 100]),xlabel('学生'),ylabel('期末成绩'),title('2005 年 12 月期末考试')
```

它产生的图象如图 3-31 所示。

我们也能用二维条形图把数据显示出来，这要通过调用 *bar*(*x*, *y*)函数。这是相对简单和直接的过程。使用 *bar*(*x*, *y*)绘制带有标签和标题图象的命令是：

```
>> x = [1:5];
>> y = [50,98,75,80,98];
>> bar(x,y), xlabel('学生'),ylabel('分数'), title('期末测试')
```

结果如图 3-32。

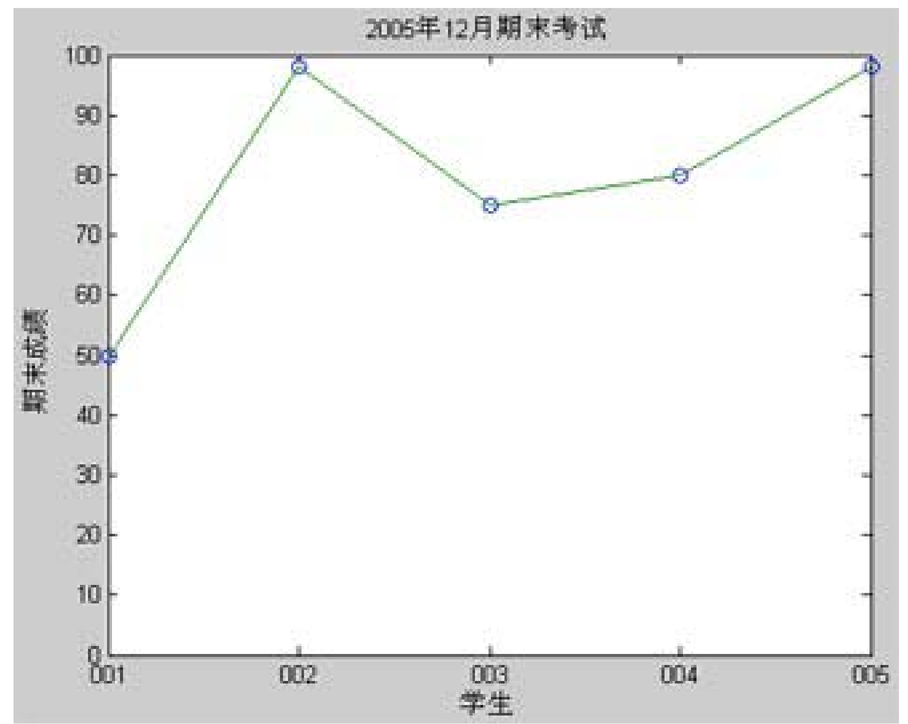


图 3-31 用 set 命令添加标签的学生测试成绩图象

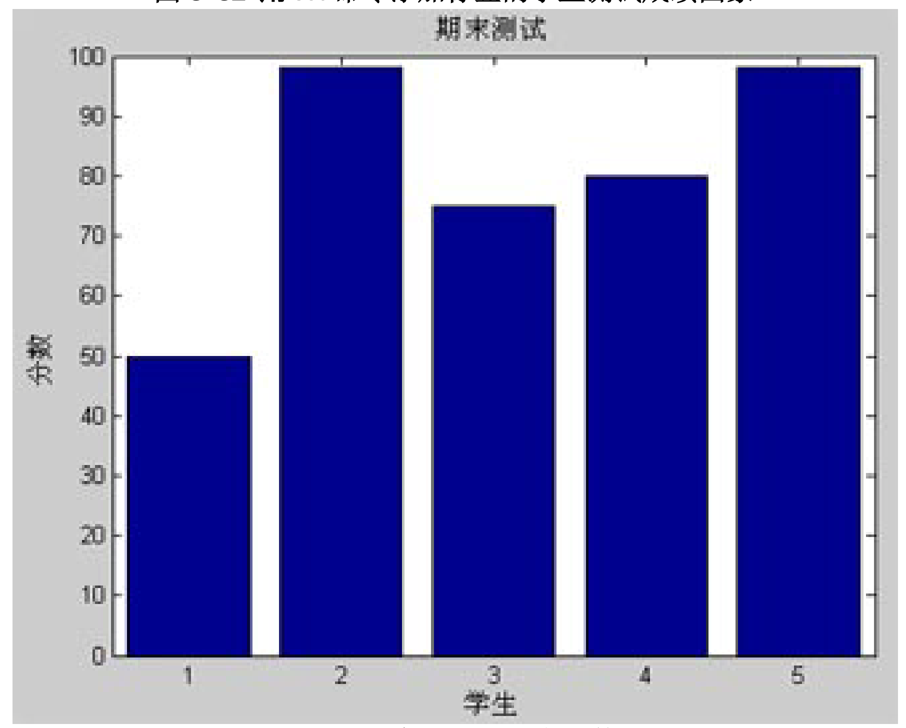


图 3-32 用条形图显示考试数据

绘制离散数据的另一种有用的方式是针状图。针状图是用函数的某些特定数据来绘制的图象，在每个点上都有一根条从水平轴或 x 轴延伸到该点，并且这些点用选择记号标示。作为例子，我们看看函数 $f(t) = e^{-\beta t}\sin(t/4)$，其中 β=0.01，并假设它是某根弹簧在某个力作用下的响应。我们先绘出 200 秒内的连续图象：

```
>> t = [0:0.1:200];
>> f = exp(-0.01*t).*sin(t/4);
>> plot(t,f),xlabel('时间(秒)'),ylabel('弹簧响应')
```

图象如图 3-33，显示了弹簧的衰减振荡情况。

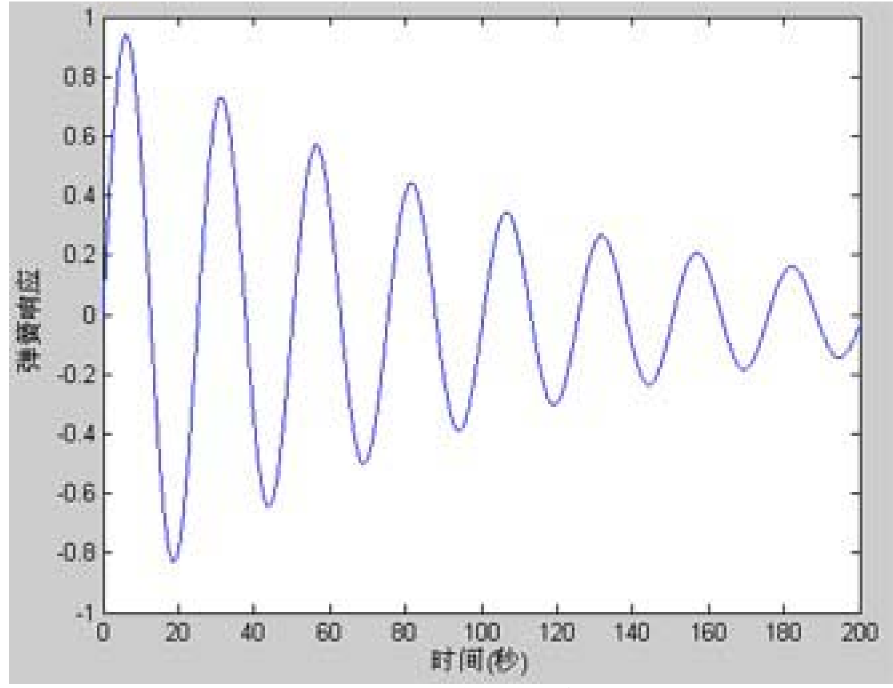


图 3-33 衰减振荡图象

现在假设我们要从离散数据绘制图象。我们将每 5 秒对系统采样一次，首先用采样时间作为数据，因此只需简单地创建一个步长为 5 秒的数组：

```
>> t = [0: 5: 200];
```

如果你已经做过实验或在计算机上仿真过，那么你会得到一组代表着每次采样的时间值。为了使用已知的数据仿真，我们简单地重新定义函数然后绘图：

```
>> f = exp(-0.01*t).*sin(t/4);
>> plot(t,f),xlabel('时间(秒)'),ylabel('弹簧响应')
```

这次我们得到的图象有些粗糙，如图 3-34。

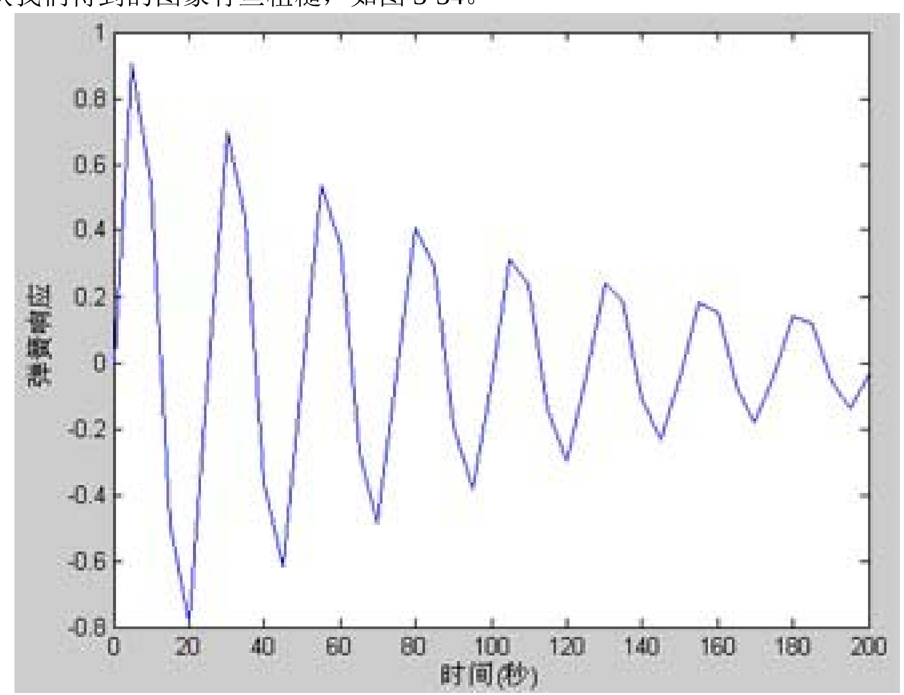


图 3-34 用较大的采样间隔绘制的图 3-33 函数的图象

对于这种情况，工程师们喜欢绘制成针头图，只需在 **MATLAB** 用命令 *stem*(*x*, *y*)即可。在例中我们使用的命令是：

```
>> stem(t,f),xlabel('时间(秒)'),ylabel('弹簧响应')
```

图象如图 3-35 所示。

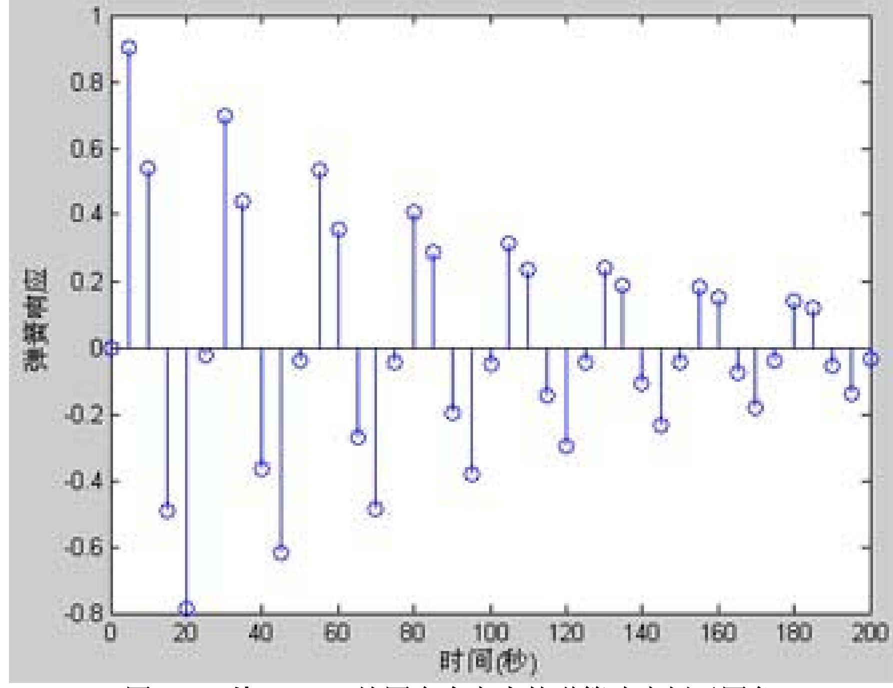


图 3-35 使用 stem 绘图命令产生的弹簧响应例子图象

*plot(x, y)*使用的线条类型选项也能够应用到 *stem* 上。例如，我们可以让 **MATLAB** 使用产生点线或虚线，或者为线条选择喜欢的颜色（默认为蓝色）。我们还可以通过向 *stem(x, y)*传递*'fill'*参数选项让 **MATLAB** 填充标记。我们还可以自由地选择标记的样式，包括方块(*s*)、菱形(*d*)、五角星(*p*)、圆圈(*o*)、叉号(*x*)、星号(*)和点号(.)。我们试试使用填充菱形标记，线条用绿色虚线。通过在 *stem* 的输入参数中传递*'--g'*告诉 **MATLAB** 使用绿色虚线。这样重新绘制的图象如图 3-36。命令是：

```
>> stem(t,f,'--dg','fill'),xlabel('时间(秒)'),ylabel('弹簧响应')
```

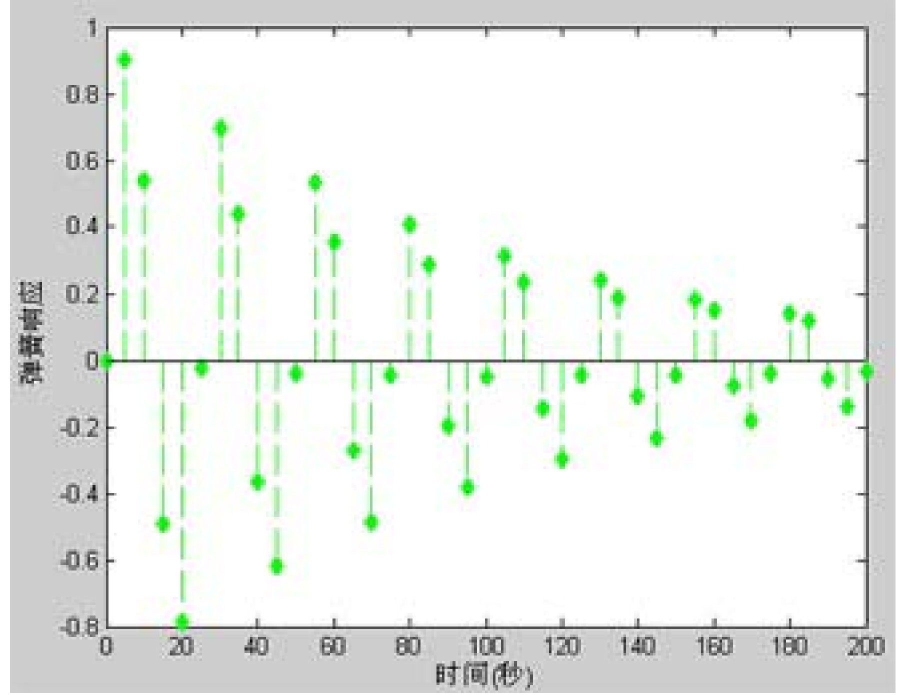


图 3-36 使用绿色填充菱形标记和虚线的针状图

等高线图

现在我们看看如何用 **MATLAB** 来产生更复杂的图象。我们以绘制等高线图来开始这一节。最简单的等高线图是仅仅绘制给定函数的等高线，而不为图象填充颜色。这可以用某些带有两个变量的函数($z = f(x, y)$)绘制得到。与往常一样，第一步是产生独立变量，这次带有两个变量 x 和 y 的集合。*meshgrid* 是一个可以为我们建立独立变量的易用函数，它所做的工作是为我们产生矩阵元素，元素 x 和 y 按照我们分别所指定的范围和增量来产生。因此，假设我们要某个函数 $z = f(x, y)$在$-5 \leq x \leq 5$ 和$-3 \leq y \leq 3$ 范围内的图象，并且增量我们都取 0.1。下面的简单调用就可以了：

```
>> [x,y] = meshgrid(-5:0.1:5, -3:0.1:3);
```

下一步，输入我们的函数。对于第一个例子，我们试用一个等高线为圆的简单函数。设 $z = x^2 + y^2$。输入如下：

```
>> z = x.^2 + y.^2;
```

现在我们调用 *contour* 命令：

```
>> contour(x,y,z)
```

产生的图象如图 3-37 所示。

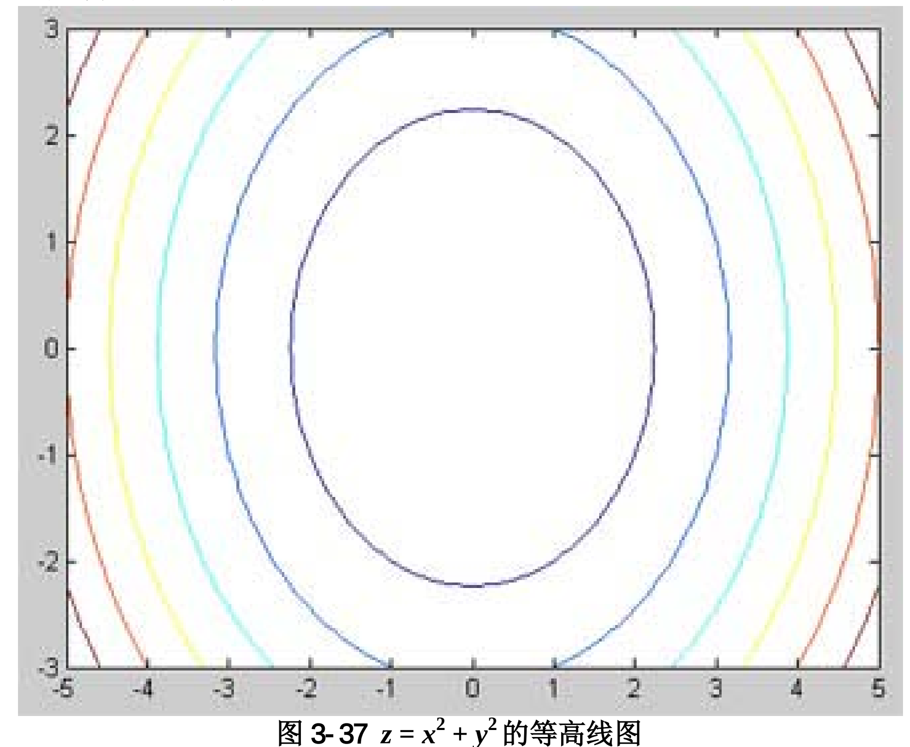


图 3-37 $z = x^2 + y^2$ 的等高线图

好，这是个好的开端，但你看过的很多图象都要比这个图象所含的信息要多一些。经常使用是为等高线图加上说明标签，告诉每条等高线之间的固定值，这可以通过调用 *set* 命令做到。首先我们要添加后面要给等高线图引用的标签，输入如下：

```
>> [C,h] = contour(x,y,z);
```

确保在行尾添加分号，这样才不会输出数据。现在我们调用 *set* 命令：

```
>> set(h,'ShowText','on','TextStep',get(h,'LevelStep')*2)
```

这一次，**MATLAB** 为每条曲线添加标签了，如图 3-38。

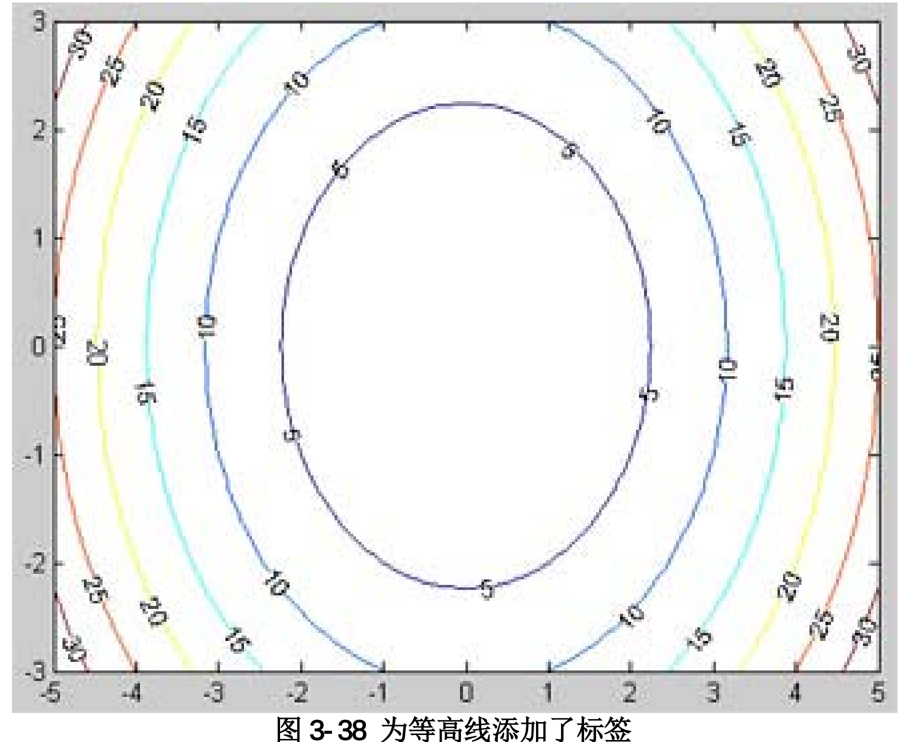


图 3-38 为等高线添加了标签

让我们再试试 $z = \cos(x)\sin(y)$。我们使用的命令是：

```
>> z = cos(x).*sin(y);
>> [C,h] = contour(x,y,z);
>> set(h,'ShowText','on','TextStep',get(h,'LevelStep')*2)
```


产生的图象如图 3-39。

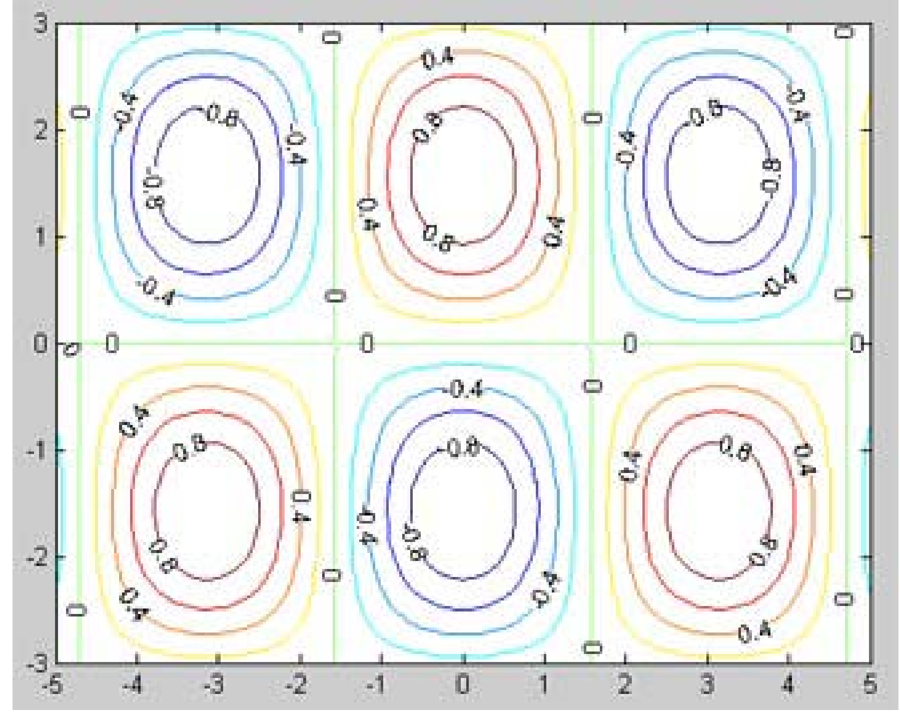


图 3-39 y = cos(x)sin(y) 在 $-5 \leqslant x \leqslant 5$ 和 $-3 \leqslant y \leqslant 3$ 范围内的等高线图

我们还可以通过调用 *contour3* 命令把等高线画成三维的。如果我们调用 *contour3(z, n)*，那么它将产生有 n 个级别的等高线。我们继续使用函数 $z = \cos(x)\sin(y)$，语句

```
>> contour3(z,10)
```

产生的图象如图 3-40。

这很有帮助，但可能还没有达到我们要求。让我们把图象变得更专业，告诉 **MATLAB** 为图象添加多一些信息。这一次我们考虑 $z = ye^{-(x^2+y^2)}$ 在 $-2 \leq x,y \leq 2$ 范围内的图象。由于变量在同一个数据范围内，使用 *meshgrid* 命令很方便：

```
>> [x,y] = meshgrid(-2:0.1:2);
>> z = y.*exp(-x.^2 - y.^2);
```

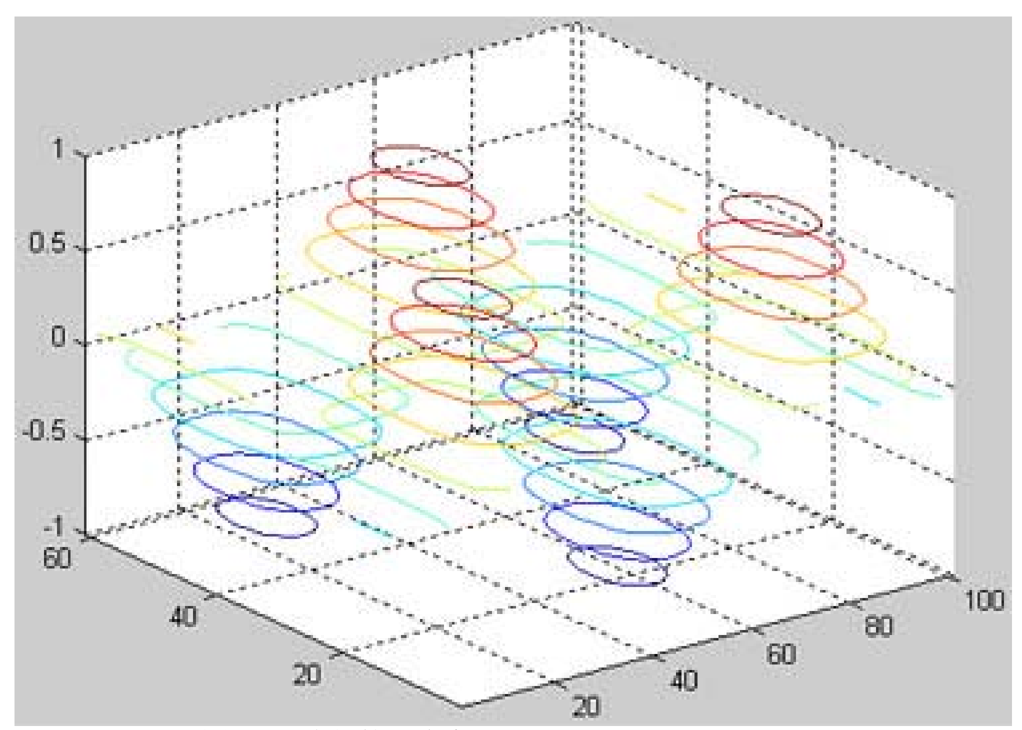


图 3-40 我们第一次为 $z = \cos(x)\sin(y)$ 调用 *contour3*

也就是，当两个独立变量在同一个范围内，你可以把 *meshgrid* 定义为[*x*, *y*] = *meshgrid*(*x*)。现在我们绘制函数的平面等高图象：

```
>> contour(x,y,z),xlabel('x'),ylabel('y')
```

结果如图 3-41 所示。

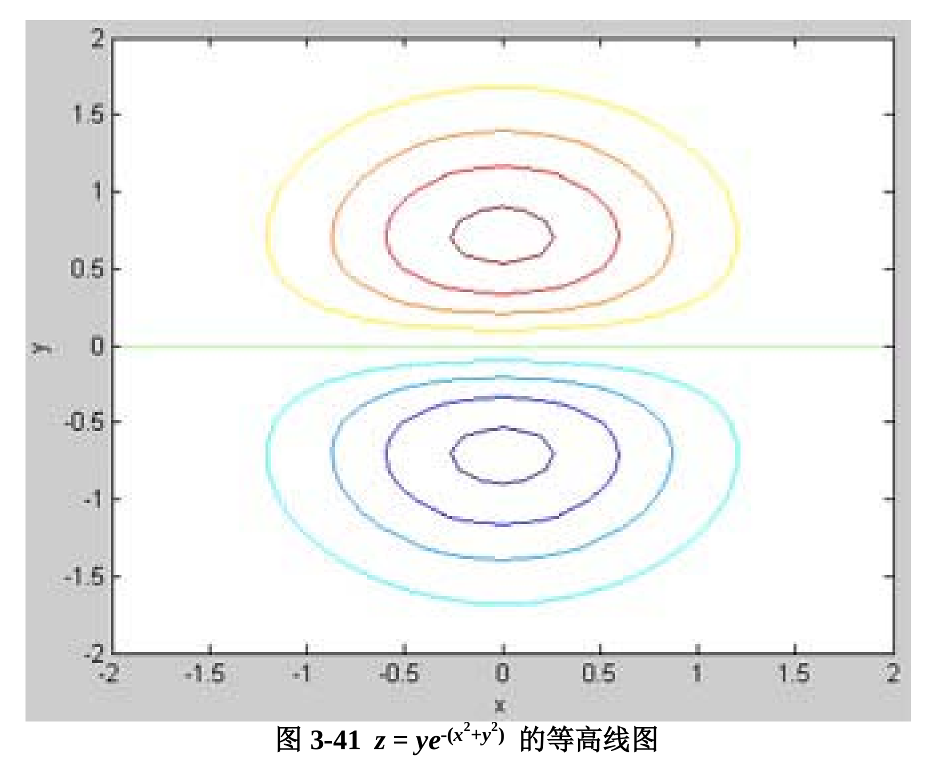


图 3-41 $z = ye^{-(x^2+y^2)}$ 的等高线图

下面，我们产生三维的等高线图。如果我们仅输入 *contour3*(*x*, *y*, *z*, 30)，那么我们得到的图象如图 3-42。

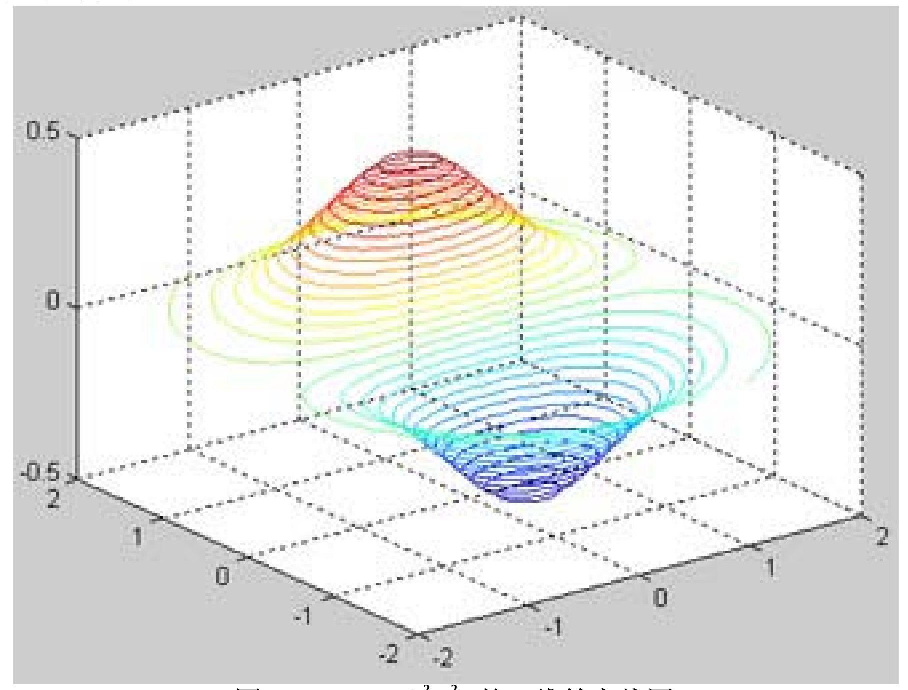


图 3-42 $z = ye^{-(x^2+y^2)}$ 的三维等高线图

通过下面的 *surface* 命令，我们可以把这个图象装扮得更漂亮：

```
>> surface(x,y,z,'EdgeColor',[.8 .8 .8],'FaceColor','none')
>> grid off
>> view(-15,20)
```

结果如图 3-43 所示，它可能曾经在你的微积分学书上出现过。

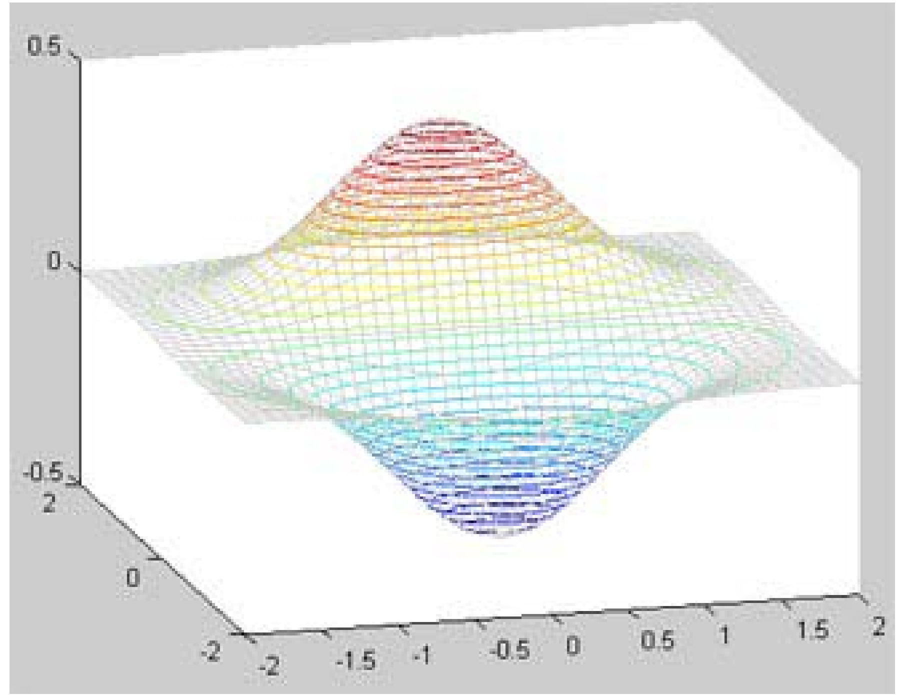


图 3-43 **使用 *surface* 命令装扮的等高线图**

三维图象

当我们使用 **MATLAB** 学习 *surface* 命令时，我们已经看到了三维绘图的一点端倪。在 **MATLAB** 中我们可以调用 *mesh*(*x*, *y*, *z*)函数产生三维图象。首先，我们看看函数 *z* = cos(*x*)sin(*y*)在 $-2\pi \leq x, y \leq 2\pi$ 内的图象。输入：

```
>> [x,y] = meshgrid(-2*pi:0.1:2*pi);
>> z = cos(x).*sin(y);
>> mesh(x,y,z),xlabel('x'),ylabel('y'),zlabel('z')
```

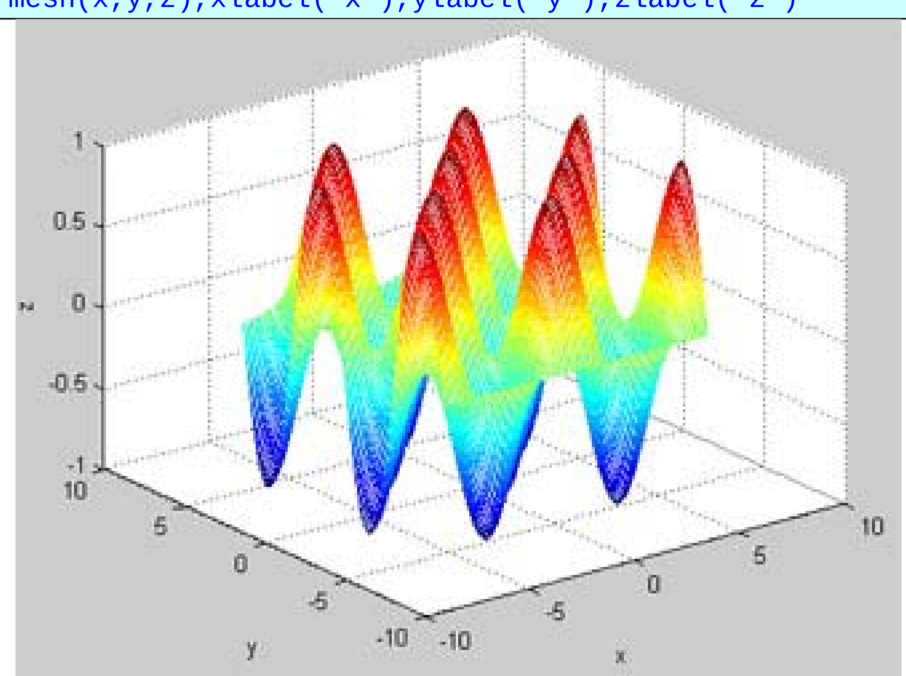


图 3-44 **使用 mesh 命令绘制的 *z* = cos(*x*)sin(*y*)图象**



如果你注意到最后一行语句，你会发现 *mesh* 是 *plot*(*x*, *y*)在三维上的扩展。结果如图 3-44 所示。

让我们试试上一节中使用的$z = ye^{-(x^2+y^2)}$，取相同的区间。我们输入下面的命令：

```
>> [x,y] = meshgrid(-2:0.1:2);
>> z = y.*exp(-x.^2-y.^2);
>> mesh(x,y,z),xlabel('x'),ylabel('y'),zlabel('z')
```

这个图象如图 3-45。

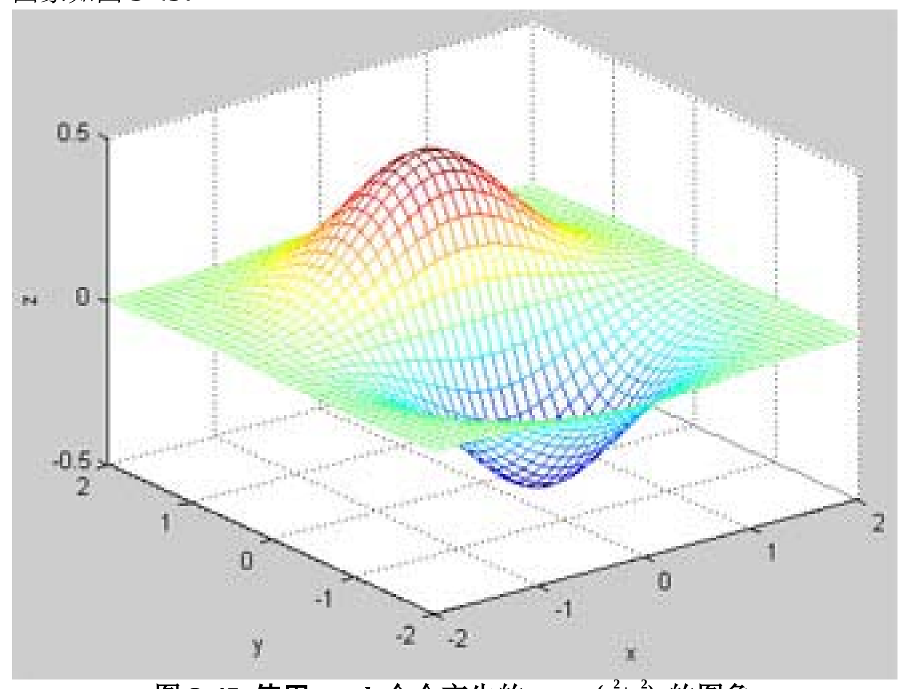


图 3-45 使用 *mesh* 命令产生的$z = ye^{-(x^2+y^2)}$ 的图象

现在我们绘制表面带有渐变颜色的图象。这可以通过 *surf* 或 *surfc* 命令做到。只需简单的更改上面例子中的命令为：

```
>> surf(x,y,z),xlabel('x'),ylabel('y'),zlabel('z')
```

图象如图 3-46 所示。

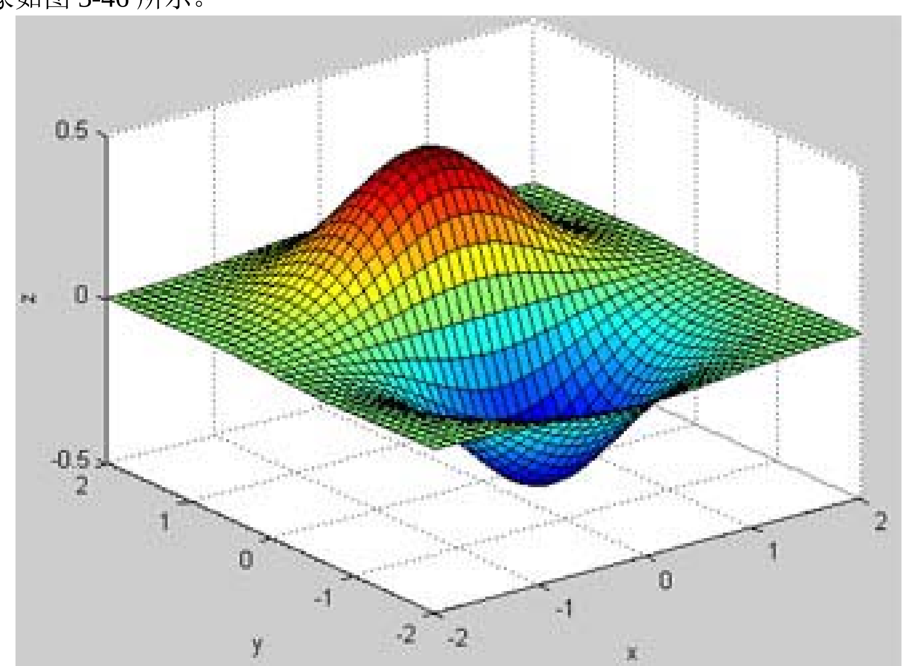


图 3-46 使用 *surf*(*x*, *y*, *z*)绘制的同一函数图象

图象表面的颜色与高度是相称的。如果使用 *surfc* 代替就会在图象中留下投影，如图

3-47。

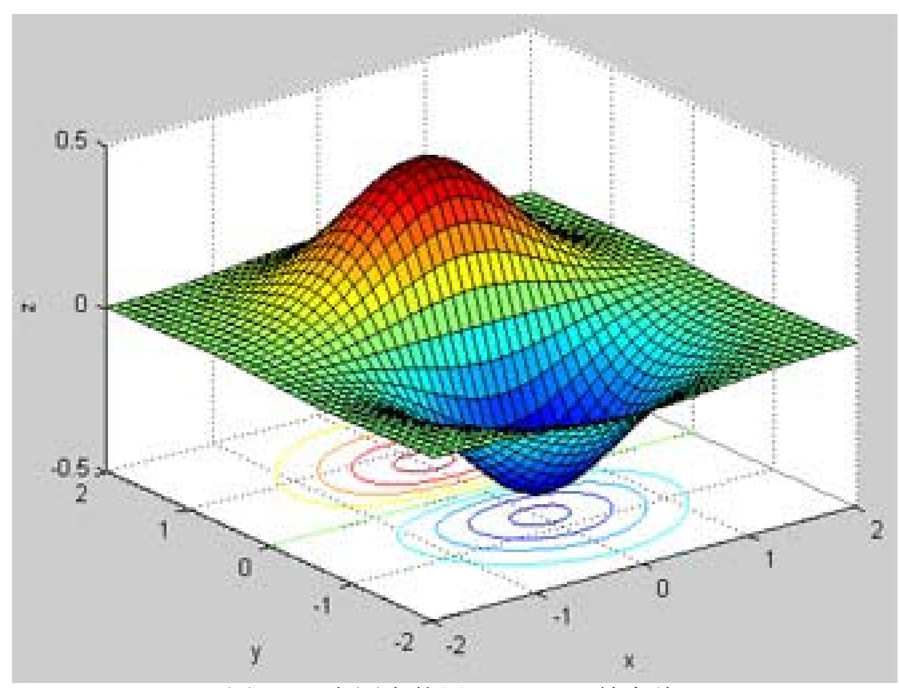


图 3-47 在图底使用 *surfc* 显示等高线

调用 *surfl*（命令中的“*l*”[*el*]告诉我们这是一个光照表面（*lighted surface*））是另一个好选择，它给了我们显示三维光照物体的表面。你可以使用这个命令产生没有线条的三维图象。图象还可以是彩色的或灰度的。例如，我们可以使用下面的命令：

```
>> surfl(x,y,z),xlabel('x'),ylabel('y'),zlabel('z');
>> shading interp;
>> colormap(gray);
```

以灰度 gray 显示的$z = ye^{-(x^2+y^2)}$ 图象如图 3-48。图象中的阴影可以设置为 *flat*、*interp* 和 *faceted*。*flat* 是用同一颜色为每个网格进行着色并隐藏网格线，而 *facted* 则显示网格，使用 *interp* 是告诉 **MATLAB** 使用颜色插值的办法进行着色，因此显得非常平滑，如图 3-48 所示（你可以在 **MATLAB** 试一下）。

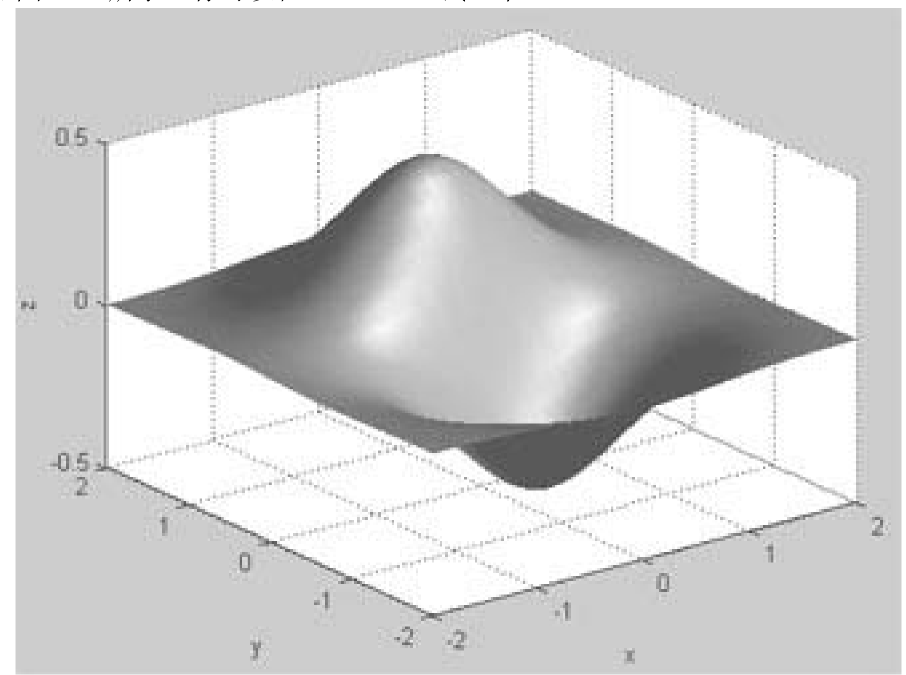


图 3-48 使用 *surfl* 绘制的函数图象

让我们产生一个有意思的类圆柱形图象。使用 **MATLAB** 内建函数可以绘制出球形或圆



柱形等基本图象。在这个例子中，我们尝试：

```
>> t = 0:pi/10:2*pi;
>> [X,Y,Z] = cylinder(1+sin(t));
>> surf(X,Y,Z);
>> axis square
```

如果我们把 *shading* 设为 *flat*，就得到图 3-49 中所示的雨滴一样、色彩鲜艳的图象。

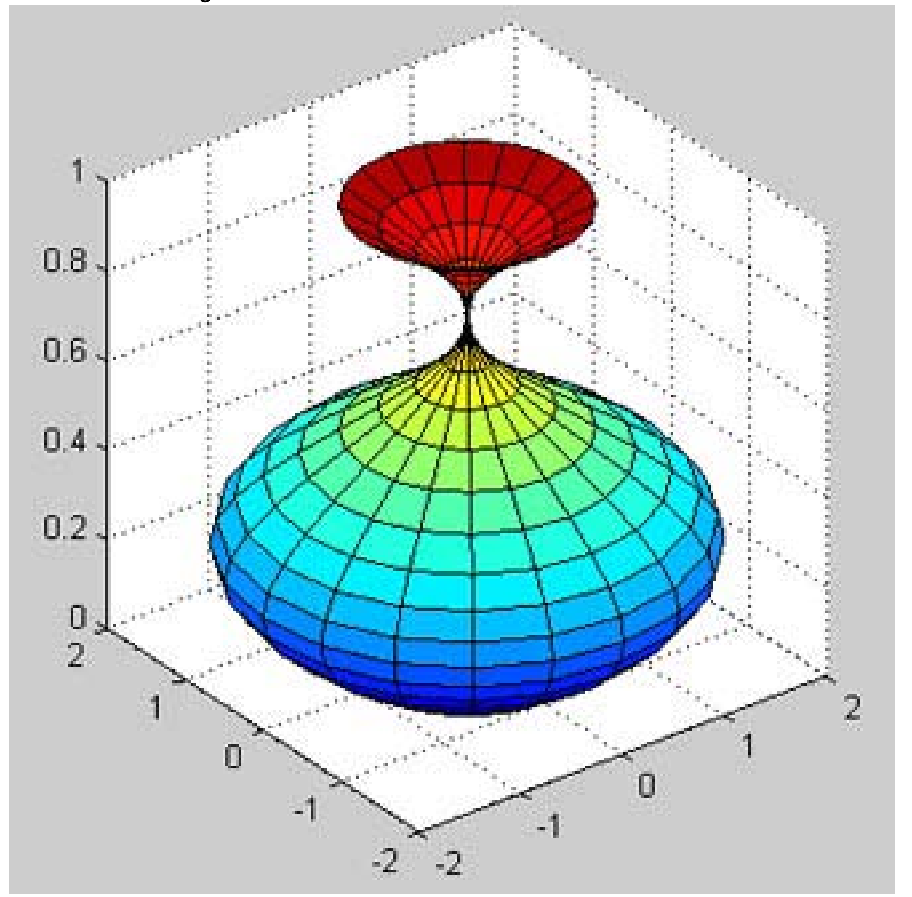


图 3-49 使用 *cylinder* 函数和 *flat* 遮光的图象

现在让我们试试另一个稍微有点不同的函数。这一次在使用 *faceted* 遮光。我们输入：

```
>> t = 0:pi/10:2*pi;
>> [X,Y,Z] = cylinder(1+cos(t));
>> surf(X,Y,Z);
>> axis square
>> shading faceted
```

MATLAB 给我们显示的图象如图 3-50。

如果我们使用 *shading interp*，那么我们得到的图象如图 3-51：

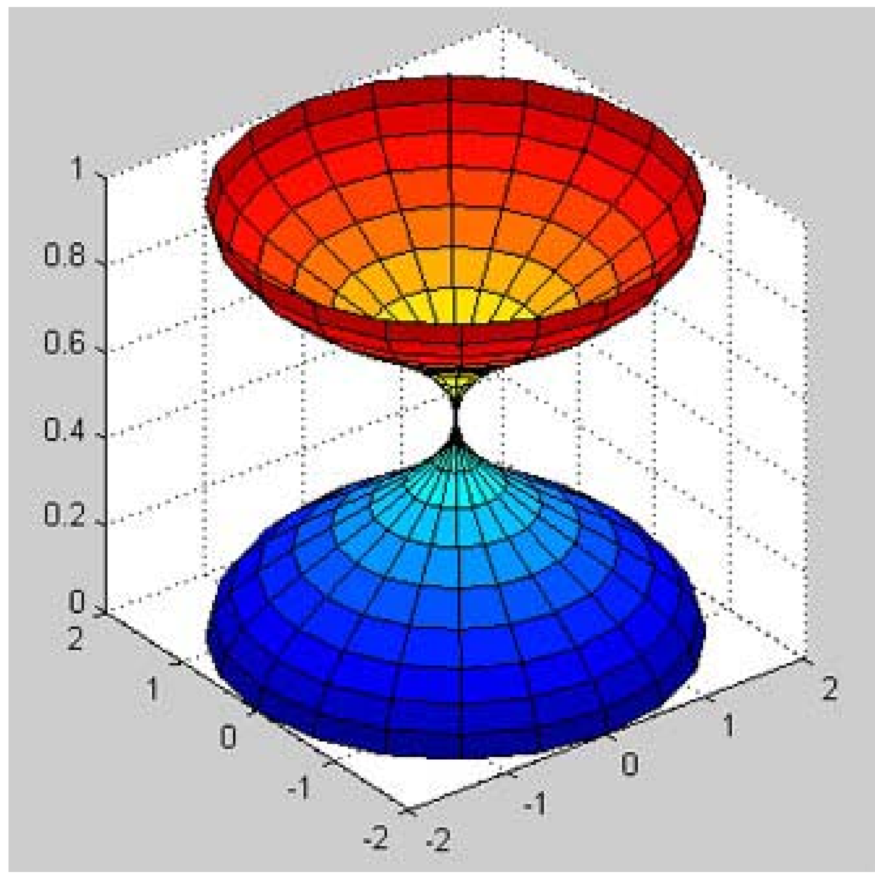


图 3-50 使用 *faceted* 遮光后 *cylinder*[1+cos(*t*)]的圆柱形图象

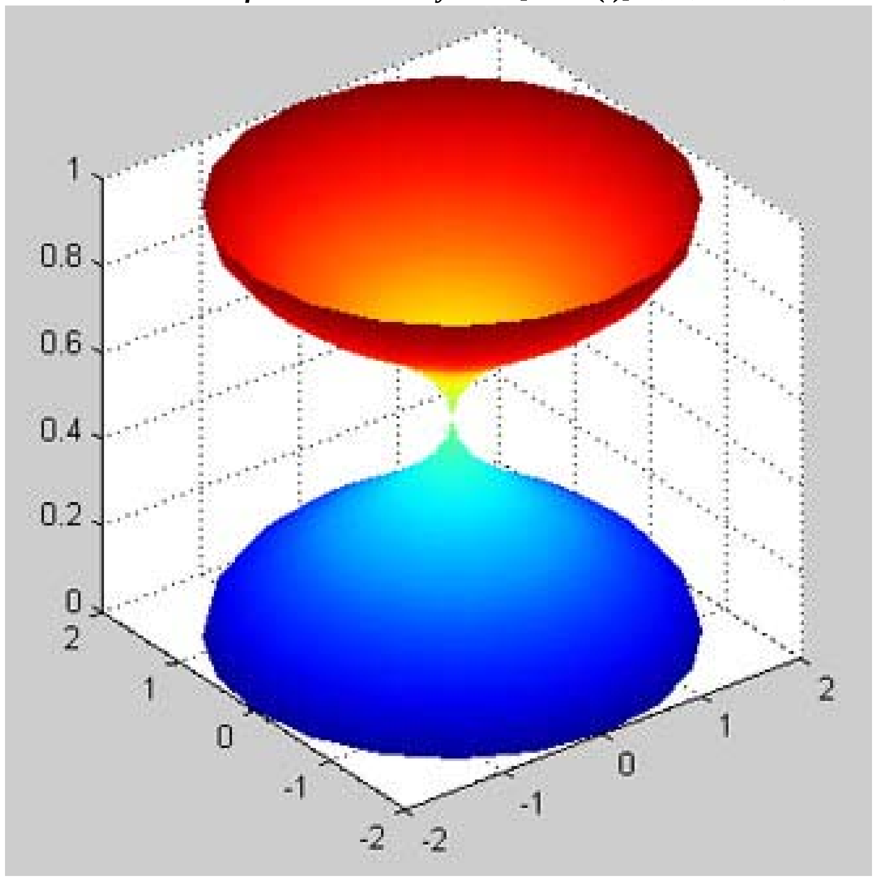


图 3-51 使用中间插值遮光的 *cylinder*[1+cos(*t*)] 图象



习题

1. 绘制正切函数 `tanx` 在 $0 \leqslant x \leqslant 1$ 上的图象，并为 x 轴和 y 轴添加标签。增量设为 0.1。
2. 显示同样的图象，把 `sin(x)` 作为第二条曲线添加到第二个图形中。
3. 生成一个列向量表示 $-\pi \leqslant x \leqslant \pi$、增量取 0.2 的数据点。分别使用 *linspace* 设置 100 个点和 50 个点再绘制图线。
4. 为一个三维图象绘制网格，取 $-3 \leqslant x \leqslant 2$ 和 $-5 \leqslant y \leqslant 5$，增量 0.1。再取 $-5 \leqslant x \leqslant 5$ 和 $-5 \leqslant y \leqslant 5$，增量为 0.2 试试。
5. 使用 *plot3* 函数绘制曲线 $x = e^{-t}\cos t$、$y = e^{-t}\sin t$ 和 $z = t$ 的图象，不要添加坐标轴标签，但要打开网格。

【参考答案在第 228 页】

第四章



统计和 MATLAB 编程介绍

MATLAB 使用起来很舒适，它能处理概率和统计学方面的问题。在本章中，我们会看到如何使用 **MATLAB** 对数据进行基本的统计、概率计算并显示结果。为了介绍 **MATLAB** 的编程工具，我们将通过编写代码解决问题来介绍，同时与使用 **MATLAB** 内置函数解决问题进行比较。



创建柱状图

对于绝大多数的情况，统计对象是一组离散数据，要计算它们的均值、平方差和标准差等。数据可以用柱状图呈现出来。我们将会用简单的例子来看如何在 **MATLAB** 做到这些。

设九年级的代数班有 36 位学生，学生在期中考中取得的分数及学生数是：

1 位学生得 100 分
2 位学生得 96 分
4 位学生得 90 分
2 位学生得 88 分
3 位学生得 85 分
1 位学生得 84 分
2 位学生得 82 分
7 位学生得 78 分
4 位学生得 75 分
6 位学生得 70 分
1 位学生得 69 分
2 位学生得 63 分
1 位学生得 55 分

在 **MATLAB** 中我们要做的第一件事是输入这些数据然后从这些数据中产生柱状图。首先我们输入分数（*x*）及得到该分数的学生数（*y*）：

```
>> x = [55,63,69,70,75,78,82,84,85,88,90,96,100];
>> y = [1,2,1,6,4,7,2,1,3,2,4,2,1];
```

通过简单调用 `bar` 命令就可产生柱状图，它就像 `plot` 一样工作，我们只需调用 `bar(x, y)`把 *x* 和 *y* 这两个数组传给它。例如，利用上面的数据，我们用下面的命令就能快速产生一个柱状图：

```
>> bar(x,y)
```

产生的柱状图如图 4-1。但是这还不够。

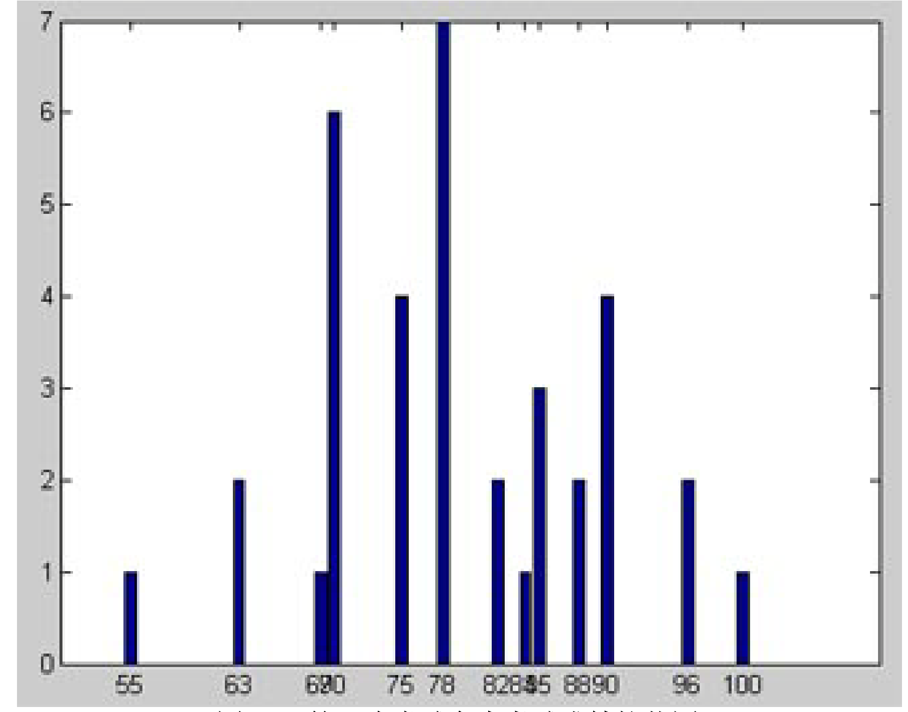


图 4-1 第一次尝试产生考试成绩柱状图

如果你是老师，你真正想知道的是可能就是到底有多少学生取得 *A*、*B*、*C* 等等。要这样做的一种方法是手动收集下面范围内的数据：

1 位学生在 50–59 分数段

3 位学生在 60-69 分数段
17 位学生在 70-79 分数段
8 位学生在 80-89 分数段
7 位学生在 90–100 分数段

接着我们用下面的方法创建两个数组，第一个数组为要包含在柱状图中的每个分数段的中间值，本例子中中间值是：

```
>> a = [54.5,64.5,74.5,84.5,94.5];
```

接着我们落在每个范围内的学生数：

```
>> b = [1,3,17,8,7];
```

现在我们再次调用 *bar(x, y)*，并添加坐标轴和标题：

```
>> bar(a,b),xlabel('分数'),ylabel('学生人数'), title('期中考代数成绩')
```

图 4-2 就是我们绘制的更专业的柱状图。

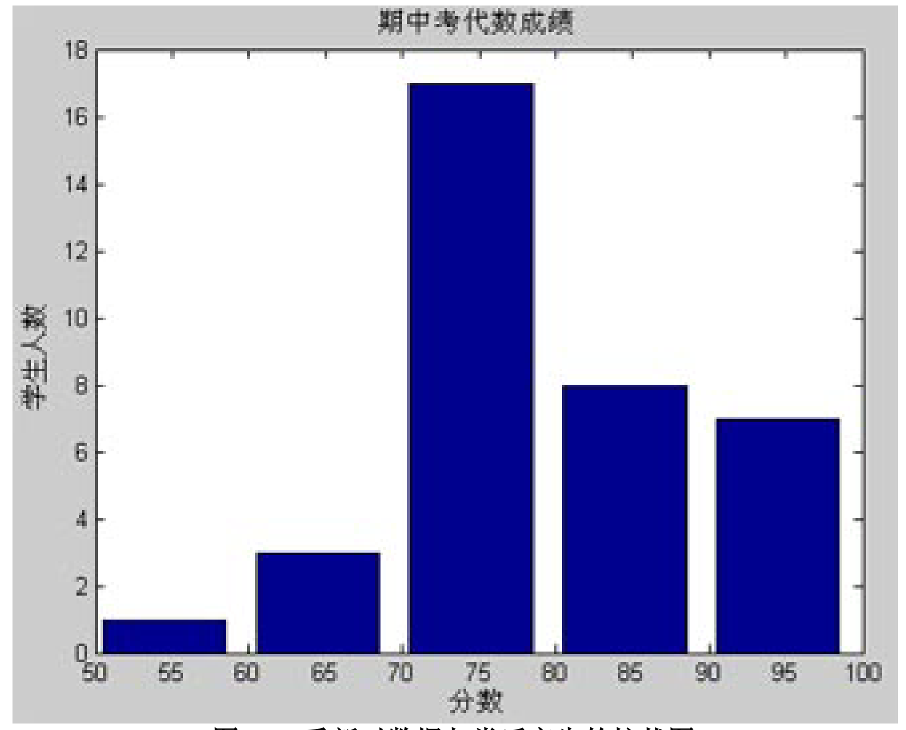


图 4-2 重新对数据归类后产生的柱状图

MATLAB 有用来产生柱状图的内置函数 *hist*。在继续之前，我们要注意有一些命令的变式也能表示数据。例如，我们可以使用 *barh* 命令产生水平的柱状图：

```
>> barh(a,b),xlabel('学生人数'),ylabel('考试分数')
```

结果如图 4-3。

你可能还想试一下使用 *bar3* 或 *bar3h* 来显示出三维图象。用下面的命令结果如图 4-4：

```
>> bar3(a,b),xlabel('考试分数'),ylabel('学生人数')
```



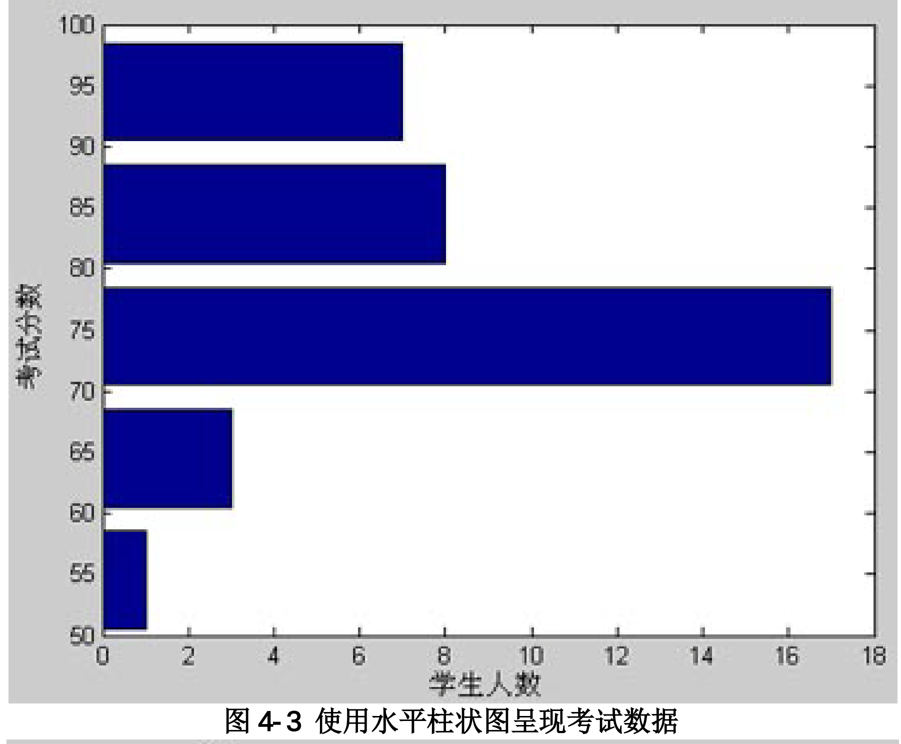


图 4-3 使用水平柱状图呈现考试数据

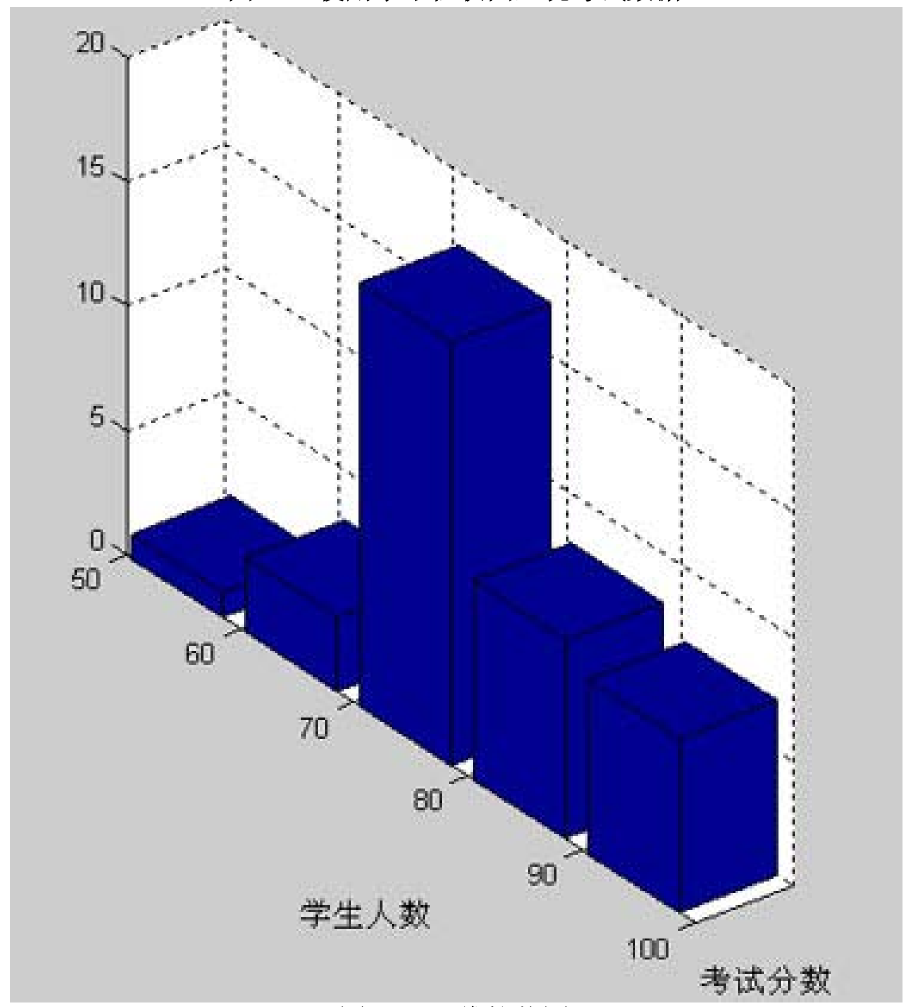


图 4-4 三维柱状图

例 4-1

中心高级中学分别由加西亚、辛普森、史密斯执教的三个代数班在期中考试中取得成绩如下：

分数范围	加西亚	辛普森	史密斯
90–100	10	5	8
80–89	13	10	17
70–79	18	20	15
60–69	3	5	2
50–59	0	3	1

对这些数据进行分类，创建组合和堆合柱状图。

解 4-1

MATLAB 中创建的带有多种数据集合的柱状图可以组合和堆合。要产生含有 *x,y* 数据的组合柱状图，我们写成 *bar(x, y, 'grouped')*，由于“*grouped*”是默认选项，因此写成 *bar(x, y)*也完全一样。要产生堆合柱状图，我们写成 *bar(x, y, 'stacked')*。输入的数据是一个多列数组，其中第一列代表加西亚班级的分数，下一列代表辛普森的，最后一列是史密斯的。首先创建含有分数范围中间值数据的 *x* 数组：

```
>> x = [54.5,64.5,74.5,84.5,94.5];
```

现在我们使用三个列向量输入分数：

```
>> garcia= [0; 3; 18; 13; 10];...
   simpson= [3; 5; 20; 10; 5];...
   smith= [1; 2; 15; 17; 8];
```

下一步是把所有的数据放进一个数组中，我们可以用下面的命令创建数组，它依次含有加西亚、辛普森和史密斯的数据：

```
>> y = [garcia simpson smith];
```

现在我们绘制柱状图：

```
>> bar(x,y),xlabel('考试分数'),ylabel('学生人数'),legend('加西亚','辛普森','史密斯')
```

结果如图 4-5 所示。

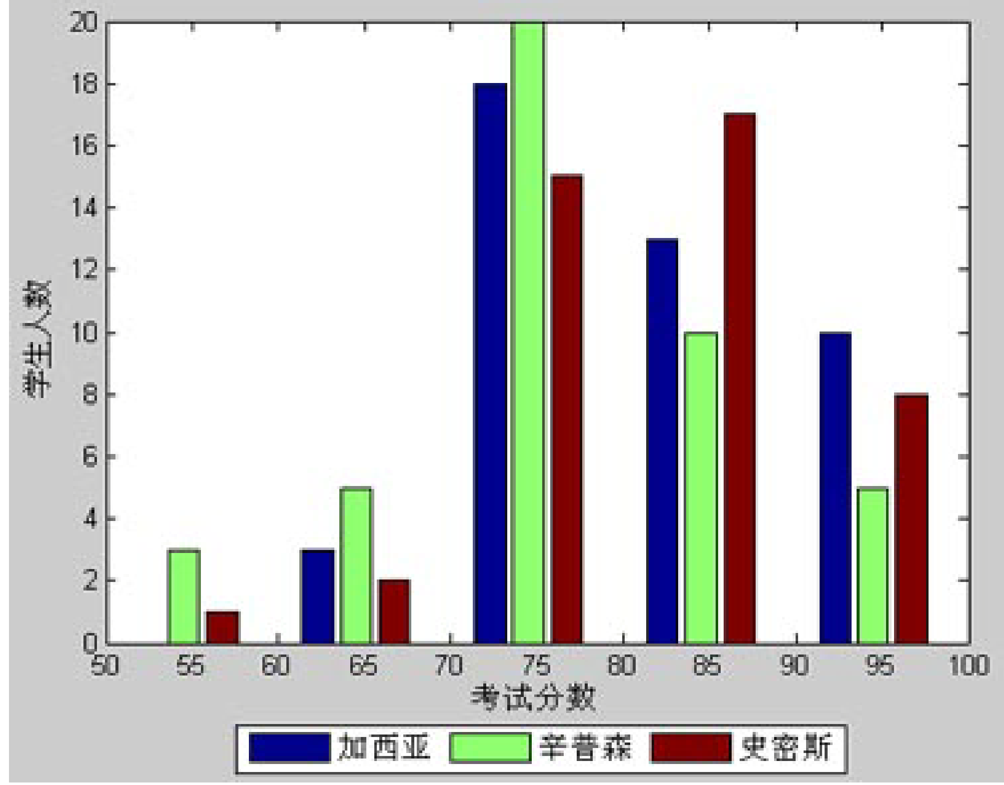


图 4-5 分组后的柱状图

基本统计

通过调用 *mean* 函数，**MATLAB** 会告诉我们一组数据的均值是多少：

```
>> a = [11,12,16,23,24,29];
>> mean(a)
ans =
   19.1667
```

我们也可以传递数组给 *mean*，**MATLAB** 会告诉我们每一列的均值：

```
>> A = [1 2 3; 4 4 2; 4 2 9]
A =
     1     2     3
     4     4     2
     4     2     9
>> mean(A)
ans =
    3.0000    2.6667    4.6667
```

不过这个简单的内置函数不能处理加权数据，要处理加权数据得手动计算均值。数集 x_j 的均值计算公式为：

$$<x_j> = \frac{\sum_{j=1}^{N} x_j N(x_j)}{\sum_{j=1}^{N} N(x_j)}$$

其中 $N(x_j)$ 是 x_j 值的个数。我们使用例子来说明——回到前面的考试分数。

为了弄清楚如何做基本统计，我们回到第一个例子。我们已经有了考试的分数集合(x)和每个分数的学生人数(y)：

```
>> x = [55, 63,69,70,75,78,82,84,85,88,90,96,100];
>> y = [1,2,1,6,4,7,2,1,3,2,4,2,1];
```

在这个例子中，x 数组代表 x_j 而 y 数组代表 $N(x_j)$。总和为：

$$\sum_{j=1}^{N} N(x_j)$$

我们只需简单地把 y 数组中的元素加起来：

```
>> N = sum(y)
N =
    36
```

现在我们用下面的公式计算总和：

$$\sum_{j=1}^{N} x_j N(x_j)$$

这是 x 和 y 数组的向量乘积。我们用 s 来代表这个中间和：

```
>> s = sum(x.*y)
s =
       2847
```

平均分数是：

```
>> ave = s/N
ave =
   79.0833
```

由于数据是以两个数组输入，我们可以计算不同分数的概率。x_j 的概率是：

$$P(x_j) = \frac{N(x_j)}{N}$$

例如，学生获得 78 分的概率，也即数组中的第六个元素的概率是：

```
>> p = y(6)/N
p =
    0.1944
```

我们可以根据概率的定义把均值重写为：

$$<xj> = \frac{\sum_{j=1}^{N} x_j N(x_j)}{\sum_{j=1}^{N} N(x_j)} = \sum_{i=1}^{N} x_j p(x_j)$$

首先，我们创建概率的数组：

```
>> p = y/N
p =
  Columns 1 through 10
    0.0278    0.0556    0.0278    0.1667    0.1111    0.1944    0.0556    0.0278    0.0833    0.0556
  Columns 11 through 13
    0.1111    0.0556    0.0278
```

现在均值就可以用下面语句计算了：

```
>> ave = sum(x.*p)
ave =
   79.0833
```


编写 MATLAB 函数

前面我们已经几次使用 *sum* 对一些基本的数进行计算了，在这里进一步介绍一些基本的 **MATLAB** 编程知识。现在我们编写一个程序（函数），利用它计算一组加权数据的均值。我们采用下面的公式：

$$\frac{\sum_{j=1}^{N} x_j N(x_j)}{\sum_{j=1}^{N} N(x_j)}$$

想要创建可以在命令窗口中可以调用的函数，第一步是创建一个.m 文件。要打开文件编辑器，我们使用下面的两个步骤：

1. 点击文件（File）下拉菜单
2. 选择新建（New）→M 文件（M-File）

打开文件编辑器，你可以在其中输入你的脚本了。行号会在窗口左边给出。第一行中，



我们要依次输入单词 *function*、用来返回数据的变量名、函数名和用来传递数据的参数。我们把函数名定为 *myaverage*。这个函数带有两个参数：

- 表示分数的数组 *x*
- 表示个数的数组 *N(x)*

我们使用变量 *ave* 返回结果。因此代码的第一行如下：

```
function ave = myaverage(x, N)
```

要正确地计算均值，*x* 和 *N* 的元素个数必须相同。我们使用 *size* 命令检查每个数组元素的个数，把结果储存到两个变量 *sizex* 和 *sizeN*：

```
sizex = size(x);
sizeN = size(N);
```

变量 *sizex* 和 *sizeN* 实际上是一个带有两个元素的行向量。例如，如果 *x* 带有四个数，那么：

```
sizex =
    1     4
```

因此要用它的值看看它们是否相等，我们必须检查 *sizex*(2)和 *sizeN(2)*。其中一种方法就是使用 *if* 语句，看看 *sizex* 是否比 *sizeN* 大或者 *sizeN* 比 *sizex* 大。在 **MATLAB** 中我们使用“管”字符“|”来表示**逻辑或**（OR）。因此用下面的语句来检查条件是合法的：

```
if (sizex(2) > sizeN(2)) | (sizex(2) <sizeN(2))
```

另外一种方法是简单地判断 *sizex* 和 *sizeN* 不相等。我们在等号“=”前面加上否定号“~”表示“不相等”，换句话说，如果 *sizex* 与 *sizeN* 不相等（NOT EQUAL）将写成：

```
if sizex(2) ~= sizeN(2)
```

如果两个数组大小不相等，我们就结束函数。如果相等则继续，然后计算均值，这时可以使用 *if – else* 语句。如果两数组大小不相等，我们所要做的就是使用 *disp* 命令在屏幕上打印出错误信息。因此 *if – else* 语句的第一部分是：

```
if sizex(2) ~= sizeN(2)
    disp('错误：数据必须具有相同的维数。')
```

注意我们已经使用符号“~=”表示不相等了。

在语句的其余部分，我们编写代码计算均值。首先，我们对取样点的值求和，即是：

$$\sum_{j=1}^{N} N(x_j)$$

在 **MATLAB** 中，我们把 *N* 作为参数传递给 *sum* 命令进行求和。因此函数接下来的两行是：

```
else
    total = sum(N);
```

现在我们再添加一行，用来计算均值公式中的分子值：

$$\sum_{j=1}^{N} x_j N(x_j)$$

这一行就是：

```
s = x.*N;
```

最后，我们计算均值，然后把它赋给声明过的变量 *ave*：

```
ave = sum(s)/total;
```

end 语句用来结束 *if - else* 语句块。因此整个函数看起来是：

```
function ave = myaverage(x,N)
sizex = size(x);
sizeN = size(N);
if sizex(2) ~= sizeN(2)
    disp('错误：数据必须具有相同的维数。')
else
    total = sum(N);
    s = x.*N;
    ave = sum(s)/total;
end
```

一旦函数编写完成，保存它以便在命令窗口中使用。**MATLAB** 会把.m 文件保存到工作文件夹。

我们回到命令窗口，看看如何使用这个函数。

假设在地方法律办公室中，雇员的年龄如下：

年龄	人数
20	2
25	3
38	4
43	2
55	3

我们创建一个表示年龄的数组 *age*，它与我们函数中的 *x* 相对应：

```
>> age = [20, 25, 38, 43, 55];
```

下一步我们创建一个称为 *num* 的数组，与我们函数中的 *N* 相对应：

```
>> num = [2, 3, 4, 2, 3];
```

调用函数，计算出平均年龄为：

```
>> myaverage(age,num)
ans =
    37
```

在继续学习之前，我们检验一下函数，确保传递的数组大小不等时，它会报告错误信息。我们尝试下面的情况：

```
>> a = [1,2,3];
>> b = [1,2,3,4,5,6];
```



当我们调用 *myaverage* 时，会得到错误信息：

```
>> myaverage(a,b)
错误：数据必须具有相同的维数。
```

使用 *for* 循环编程

***for* 循环**是一个指令，它告诉 **MATLAB** 对围起来一段语句执行一定量的次数。*for* 循环是语法是：

```
for index = start : increment : finish
  statements
end
```

我们通过写一个简单的函数来理解 for,这函数对列向量或行向量的所有元素进行累加。如果我们把增量参数省略，**MATLAB** 就认为我们把增量设为 1。

在我们的函数中，第一步是声明函数名称和获取传递进来的数组大小：

```
function sumx = mysum(x)
%获取元素个数
num = size(x);
```

这里我们添加了一个新的编程元素——我们包含了一行注释。注释是为读者准备的说明性语句，会被 **MATLAB** 忽略。在一行的开头放置%就表示它是注释。现在我们创建一个用来保存总和的变量，并把它初始化为零。

```
%初始化总和
sumx = 0;
```

现在我们使用 for 循环遍历列量中的每个元素一次。

```
for i = 1:num(2)
  sumx = sumx + x(i);
end
```

在写函数时，确保每行语句后面加上分号“;”——除非你想把结果显示在屏幕上。

计算标准差和中位数

现在让我们使用 **MATLAB** 的基本统计分析工具计算一组离散数据的标准差和中位数。假设数据是以频数给出或已知数据点数量。作为举例，我们再次使用办公室雇员的例子，设每个年龄及相应的数量是：

2 位雇员的年龄是 17
1 位雇员的年龄是 18
3 位雇员的年龄是 21
1 位雇员的年龄是 24
1 位雇员的年龄是 26
4 位雇员的年龄是 28
2 位雇员的年龄是 31
1 位雇员的年龄是 33
2 位雇员的年龄是 34
3 位雇员的年龄是 37

1 位雇员的年龄是 39
2 位雇员的年龄是 40
3 位雇员的年龄是 43

我们要做的第一件事是创建一个绝对频数数组，这就是我们在前一节中使用的数组 *N(j)*。这一次我们给每个年龄设置入口，所给年龄没有雇员的地方设为 0。我们把绝对频数称为 *f_abs*：

```
f_abs = [2, 1, 0, 0, 3, 0, 0, 1, 0, 1, 0, 4, 0, 0, 2, 0, 1, 2, 0, 0, 3, 0, 1, 2, 0, 0, 3];
```

我们要把数据“装箱”，因此我们定义一个“装箱宽度”*binwidth*。由于我们是一岁一岁度量的，故把 *binwidth* 设为 1：

```
binwidth = 1;
```

我们创建一个数组表示 17 到 43 之间的年龄，*binwidth* 是 1。

```
bins = [17:binwidth:43];
```

现在我们收集未加工的数据，使用 for 循环遍历所有数据，如下：

```
raw = [];
for i = 1:length(f_abs)
    if f_abs(i) > 0
        new = bins(i)*ones(1,f_abs(i));
    else
        new = [];
    end
    raw =[raw,new];
end
```

这个循环就是创建一个数组，按频数重复每个元素：

```
>> raw
raw =
  Columns 1 through 16
    17    17    18    21    21    21    24    26    28    28    28    28    31    31    33    34
  Columns 17 through 26
    34    37    37    37    39    40    40    43    43    43
```

现在我们可以用 **MATLAB** 内置的函数计算这些未加工数据的统计信息了。例如，雇员年龄的均值是：

```
>> ave = mean(raw)
ave =
   30.7308
```

我们可能也对中位数感兴趣，它会告诉我们哪个年龄有一半雇员比它年青，一半比它老：

```
>> md = median(raw)
md =
    31
```

标准偏差是：

```
>> sigma = std(raw)
sigma =
    8.3836
```



如果标准偏差较小，这意味着很多数据落在均值附近；如果标准差较大，那么数据就更分散。由于本例中 *bin* 大小为 1（年），标准差为 8.4（年），说明是这些数据是比较分散的。我们把数据按照频数比例绘制成柱状图象。第一步是计算数据的“面积”：

```
area = binwidth*sum(f_abs);
```

计算比例：

```
scaled_data = f_abs/area;
```

产生图象：

```
bar(bins,scaled_data),xlabel('年龄'),ylabel('频数比例')
```

结果如图 4-6。

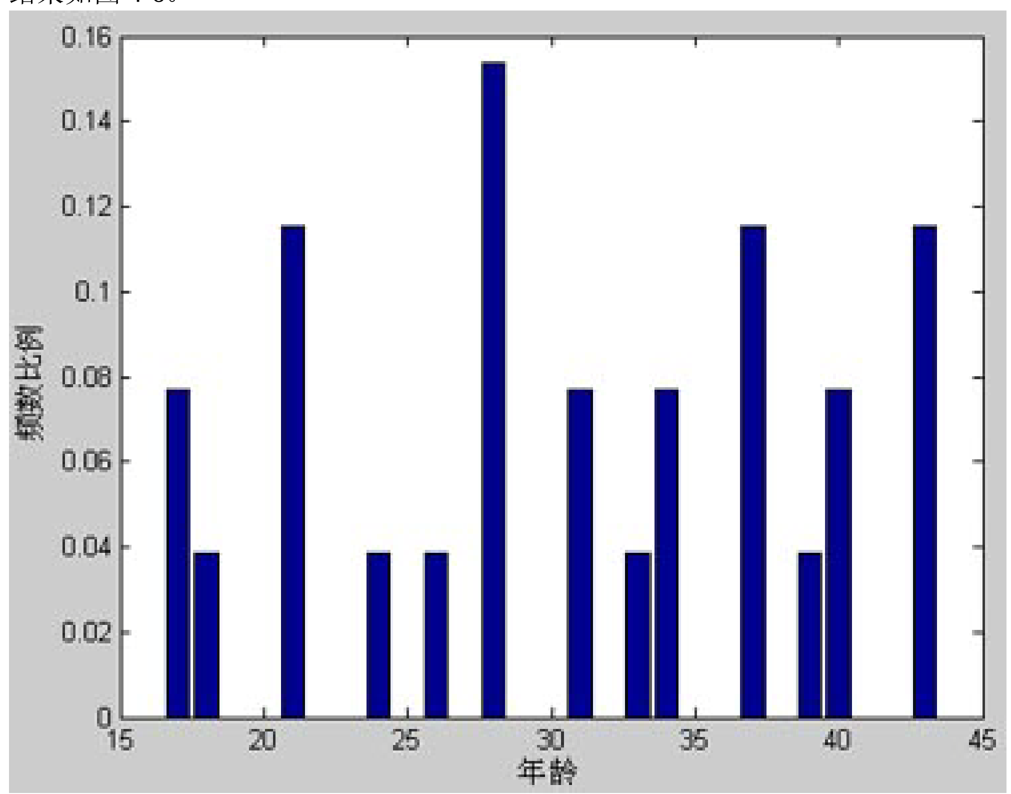


图 4-6 标准偏差较大的年龄数据

正如从图象中看到：数据并不很好地落在均值——即大约 31 岁——周围。作为对照，我们考虑另一个分布较好的办公室人员。假设雇员的年龄落在 17-34 之间，它们的频数分布如下：

```
f_abs = [2, 1, 0, 0, 5, 4, 6, 7, 8, 6, 4, 3, 2, 2, 1, 0, 0, 1];
```

如果我们按相同的办法处理，我们得到的频数比例柱状图如图 4-7 所示。

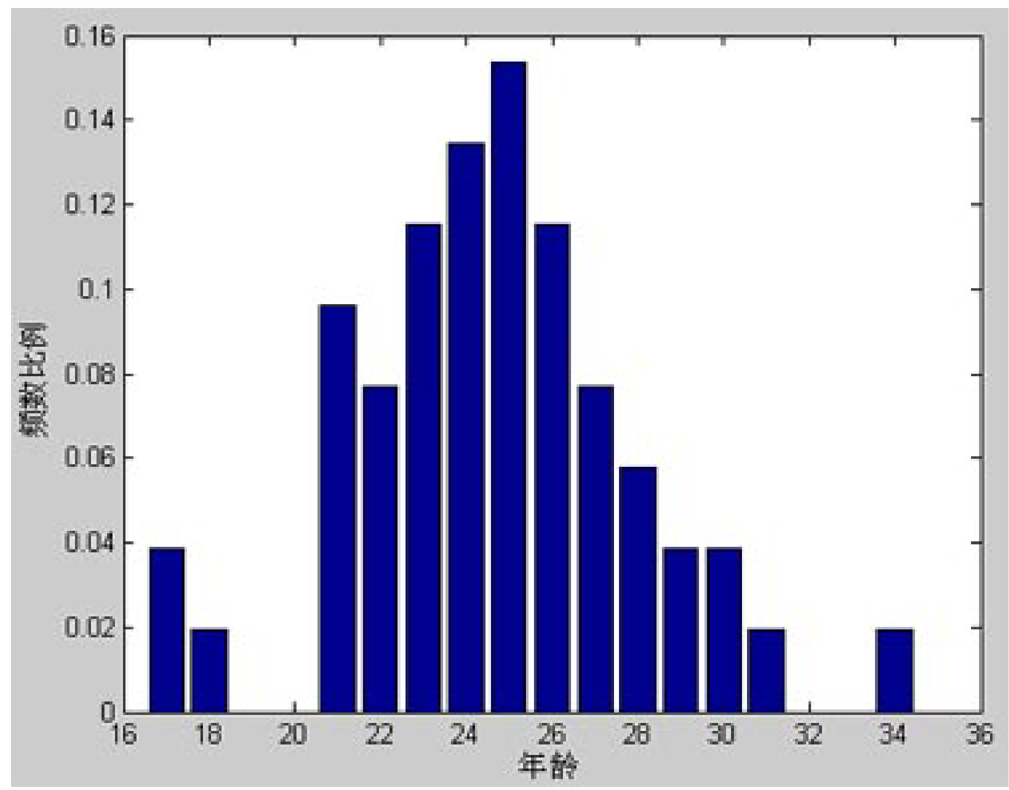


图 4-7 标准偏差较大的数据

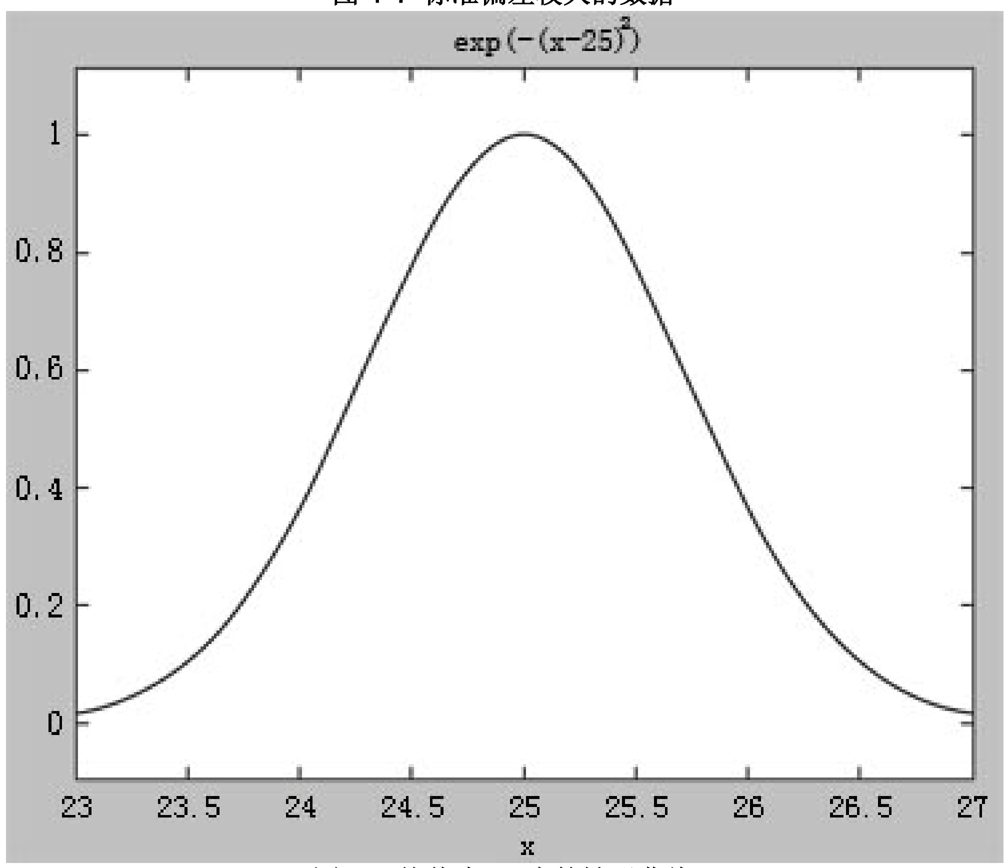


图 4-8 均值为 25 岁的钟形曲线

这组雇员的基本统计数据是：

```
>> mu = mean(raw)
mu =
```



```
    24.6538
>> med = median(raw)
med =
     25
>> sigma = std(raw)
sigma =
     3.3307
```

这个例子的标准偏差小得多——我们可以从频数比例图象中看出。它与高斯或钟形曲线很相似，如图 4-8。

当数据符合高斯分布时，标准差就可以用来描述数据，确定落在某个数据上的概率。我们可以把最后的数据看成是符合的这种情况的。标准偏差用 σ 表示，均值用 μ 表示，那么曲线落在下列范围：

$\mu - \sigma \le x \le \mu + \sigma$、$\mu - 2\sigma \le x \le \mu + 2\sigma$ 和 $\mu - 3\sigma \le x \le \mu + 3\sigma$

的百分率分别是 68%、96%和 99.7%。使用前面的年龄数据集合，我们得到它的标准差和均值是：

```
>> sigma = std(raw)
sigma =
     3.3307
>> mu = mean(raw)
mu =
    24.6538
```

大约 68%的年龄落在均值的 1 个标准偏差范围内，即是说，落在 *mu* - *sigma* = 21.3232 岁和 *mu* + *sigma* = 27.9845 岁之间。再宽一些，96%的年龄将在 *mu* - 2**sigma* = 17.9925 岁和 *mu* + 2* *sigma* = 31.3152 岁之间。注意结果并不精确，这是由于我们的数据集并不一个严密的钟形曲线，但我们演示了概念及如何使用 **MATLAB** 分析接近钟形曲线的数据。

更多编程要点

之前我们写过一个像用计算机代码写成的脚本文件(.m 文件)实现了一个功能，我们稍微离开本章的主题，了解更多一些编程的结构。首先，我们看看如何通过提示符实时得到用户或你的数据。这可以通过 *input* 函数做到。要使用 *input*，你要设置一个变量用来保存输入的数据。*input* 函数带一个用单引号引起来的字符串为参数。当执行到该语句时，**MATLAB** 在命令窗口打印出该字符串，等待用户的输入数据。当用户敲回车键后，用户所输入的数据将被正确地保存到变量中。

我们编一个简单的例子，看看如何使用。假设缅因州的某个偏僻小村的房子的价格是$10 每平方英尺，我们要得到一个房子的面积平方英尺数，通过询问房子的平方数输出总价钱。我们可以使用下面的命令询问和取得数据：

首先，由于我们在这里是处理金融数据，我们告诉 **MATLAB** 用 *bank* 格式显示结果。然后设置每平方英尺的价格。

```
>> format bank
>> rate = 10;
```

现在我们使用 *input* 命令要求用户输入房子的平方数：

```
>> sqft = input('请输入房子的总平方数：')
```

这样 **MATLAB** 会输出：

```
请输入房子的总平方数：
```

然后 **MATLAB** 耐心等待用户的输入，右边还有一光标在闪动。我们输入 1740。结果是：

```
sqft =
      1740.00
```

由于我们已经选用了 *format bank*，**MATLAB** 在计算过程中使用了小数点后两位。很不幸运，它没有友好的给我们打印出美元号“$”。

现在我们做计算：

```
>> price = rate*sqft
price =
     17400.00
```

现在缅因州的某个偏僻小村的房地产很便宜，不过这不是我们所关心的。我们如何才能把它给显示出来呢？我们使用 *disp* 命令。让我们告诉用户结果：

```
>> disp('总价钱是$：'), disp(price)
总价钱是$:
     17400.00
```

好了，到这里我们学习了两个重要的事情——如何从用户得到数据及给用户返回答案。现在我们开始看看如何控制程序的流程：

这里是另一个例子。写一个程序要求用户输入球的半径，利用它计算球的体积，并把结果输出：

```
function volume
%要求用户输入半径
r = input('输入半径：')
vol = (4/3)*pi*r^3;
disp('体积是：')
disp(vol)
```

while 语句

学过计算机科学课程的人都遇过 while 语句。在 MATLAT 中，我们输入 while 语句如下：

```
while condition
   statements
end
```

假设我们要计算 n 个数值的和，n 是由用户输入的。

在取得用户输入的条件后，我们可以用 while 循环来做。首先我们询问用户有多少个数值需要求和：



```
n = input('输入求和的个数：')
```

然后我们初始化一些变量：

```
i = 1;
sum = 0;
```

现在就是 *while* 循环：

```
while i <= n;
   sum = sum + 1/i;
   i = i + 1;
end
```

我们可以像下面一样把答案报告给用户：

```
disp('总和为：');disp(sum)
```

如果你不是程序员，你要记下的一条笔记是——确保在你的 while 循环块中对增量进行改变，本例中使用 i = i + 1 语句，否则 **MATLAB** 将会死循环。

如果我们让 n = 5 运行代码，那么 **MATLAB** 会告诉我们总和为 2.2833。

switch 语句

让语句的运行情况决定于运行时的另一种方法是使用 switch 语句。它的语法是：

```
switch expression
   case 1
      do these statements
   case 2
      do these statements
   case n
      do these statements
end
```

case 的检查条件可以是任何一种类型的变量，比如字符串或实数。如果有多个条件要执行的语句是相同的，它们就可以组合到一个 *case* 语句中，使用逗号隔开列出各种条件并用圆括号括起来。

让我们用一个例子来说明。假设在政府部门工作的工资等级有：

1、2、3、4

对应的工资是：

$40,000、$65,000、$65,000 和$85,000

我们可以使用 *switch* 语句把各等级的工资赋给 *pay*：

```
switch grade
   case 1
      pay = 40000
   case ( 2,3 )
      pay = 65000
   case 4
```

```
        pay = 85000
    end
```

习题

1. 一组男生的体重（单位为磅）如下表列出：

体重（磅）	人数
130	2
138	1
145	3
150	6
152	1
155	3
160	1
164	1
165	3
167	4
170	3
172	2
175	1

   使用 **MATLAB** 找出体重均值，中位数和标准差。
2. 使用 **MATLAB** 编写程序计算加权数据中某个给定的数的概率。使用练习 1 中的数据，学生重 150 磅的概率是多少？
3. 绘制数据的频率比例柱状图。从你的图象读出 130 磅的频率比例。
4. 编写一个 **MATLAB** 程序，要求用户输入圆柱体的半径和高，然后输出体积。
5. 用 *while* 循环计算 $1 + x + x^2 + \ldots + x^n$ 的值。
6. 用 *for* 循环编写程序计算 $1 + \frac{1}{x} + \frac{1}{x^2} + \ldots + \frac{1}{x^r}$ 的和。

【参考答案在第 228 页】

第五章



代数方程求解和其它符号工具

在本章中我们将学习使用 **MATLAB** 解方程。我们先看一个简单的只含一个变量的代数方程及解，然后我们再看看如何使用处理超越函数、三角函数和双曲函数，最后，我们学习使用 **MATLAB** 处理复数。下一章，我们再学习使用 **MATLAB** 解微分方程。

解基本代数方程

在 **MATLAB** 中，我们使用 *solve* 命令解代数方程，所要做的事是把方程用单引号引起来然后敲回车。我们从一个非常简单的例子学习起。假设我们要使用 **MATLAB** 找出下列方程的解 *x*：

$$x + 3 = 0$$

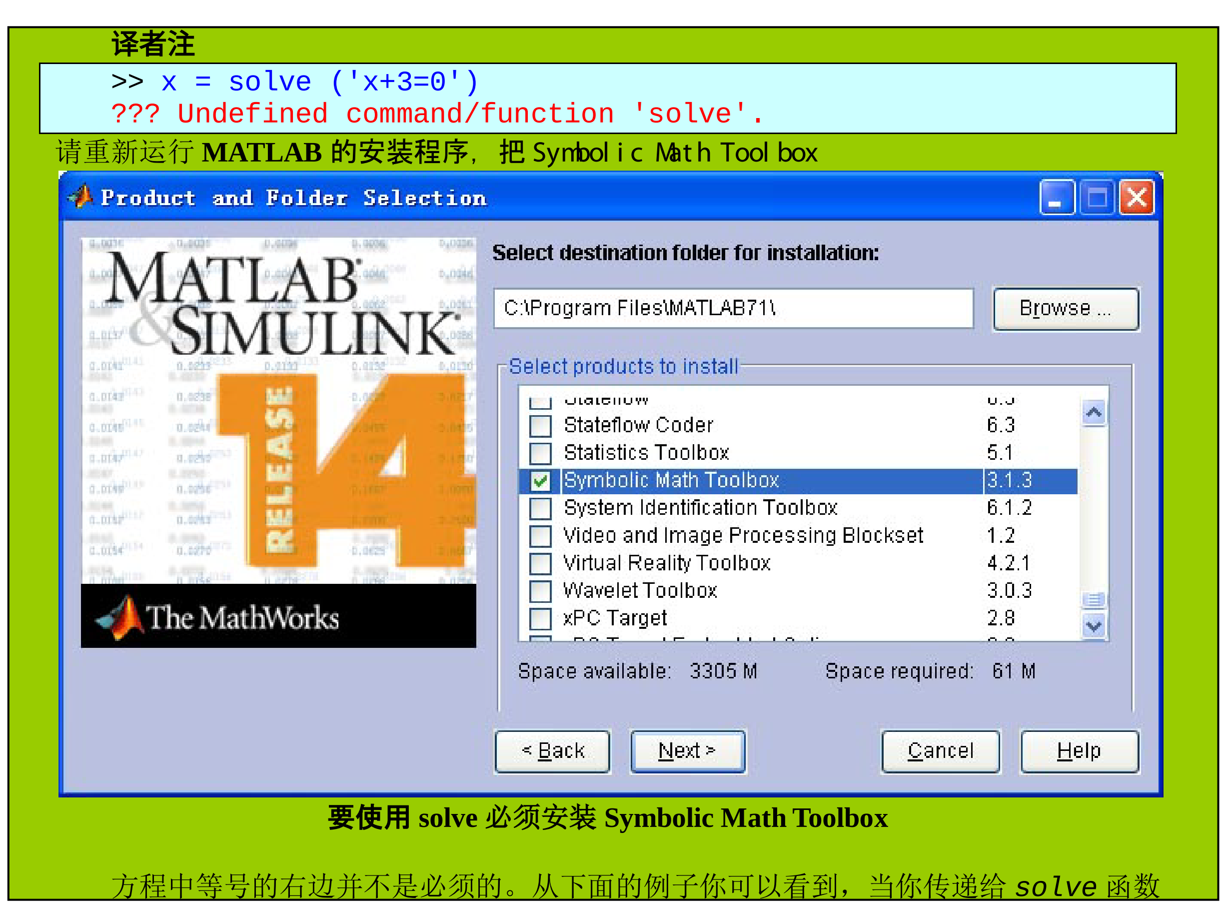
有疑问？——聪明的读者一看就知道答案是 *x* = -3，为什么我们还要为这个问题大费周折呢——原因是在这里我们可以看到使用 **MATLAB** 进行符号计算是多么地容易。我们只用一步就可以得到解。所要做的就是创建一个变量，然后把 *solve* 返回的值赋给它，如下：

```
>> x = solve('x+3=0')
x =
-3
```

译者注

```
>> x = solve ('x+3=0')
??? Undefined command/function 'solve'.
```

请重新运行 **MATLAB 的安装程序**，把 Symbolic Math Toolbox



要使用 solve 必须安装 Symbolic Math Toolbox

方程中等号的右边并不是必须的。从下面的例子你可以看到，当你传递给 *solve* 函数 *x* + 8 时，**MATLAB 假定**你传递的就是 *x* + 8 = 0。请看例子：

```
>> x = solve('x+8')
x =
-8
```

因此你可以以你喜欢的方式输入方程，对于我自己来说，我喜欢尽可能地清楚明了，因此我更喜欢使用 x + 8 = 0 作为参数。

你要求解的方程有时候可能含有多个符号。例如，我们可能会碰到像下面一样方程含有常量 a 的情况：

$$ax + 5 = 0$$



如果我们在 **MATLAB** 输入这个方程，它只假定你要求解 *x*：

```
>> solve('a*x+5')
ans =
-5/a
```

其实还有第二种方法调用 *solve*，我们可以告诉它要它解**哪个符号**。语法如下：

```
solve(equation, variable)
```

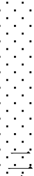
与你传递给 *solve* 的方程一样，变量 *variable* 也应该用单引号括起来。回到方程 ax + 5 = 0，让我们告诉 **MATLAB** 解出 *a* 来。输入如下：

```
>> solve('a*x + 5','a')
```

MATLAB 输出为：

```
ans =
-5/x
```

二次方程求解



学代数的学生一定会高兴：*solve* 命令还可以用来求解高阶方程。对于不接触代数很久的我们，如果突然遇到二阶或三阶方程，**MATLAB** 提供了一种方法检查我们的计算结果，或是把我们从解题的无聊中解脱出来。

使用的步骤基本上一样，我们使用脱字符号(^)指定次数。我们考虑下面的方程：

$$x^2 - 6x - 12 = 0$$

这方程我们可以手工解出来，化成平方形式或应用公式求出解。使用 **MATLAB** 求解，我们写成：

```
>> s = solve('x^2 - 6*x - 12 = 0')
```

MATLAB 返回方程的两个根：

```
s =
3 + 21^(1/2)
3 - 21^(1/2)
```

现在如何处理 *solve* 返回的结果？你可以取出它们，与其他的 **MATLAB** 变量一样使用它们。对于本例中的两个根，它们是以 *s*(1)和 *s*(2)的形式保存的。所以我们可以使用其中的一个根来定义一个新的量：

```
>> y = 3 + s(1)
y =
6+21^(1/2)
```

下面是引用 *solve* 返回的两个根的例子：

```
>> s(1) + s(2)
ans =
6
```

当 *solve* 只返回一个变量时，数组的语法就不必要了。例如，我们使用 *x* 保存下面方程的解：

```
>> x = solve('3*u + 9 = 8')
```

MATLAB 如实告诉我们：

```
x =
-1/3
```

现在我们可以在另一个方程中使用这个结果：

```
>> z = x + 1
z =
2/3
```

把一个方程赋给一个变量，然后再把它传递给 *solve* 也是完全可以的。例如，让我们随便写一个方程，然后把它赋值给 *d*：

```
>> d = 'x^2 + 9*x -7 = 0';
```

现在我们调用 *solve*：

```
>> solve(d)
```

MATLAB 正确的给出方程的两个根：

```
ans =
-9/2+1/2*109^(1/2)
-9/2-1/2*109^(1/2)
```


符号方程绘图

我们绕开解方程，看看如何对输入的材料进行绘图。好了，当我说 “x^2 - 6 * x - 12 = 0”比写成“x^2 - 6*x - 12”要好的时候，当然你也有好理由选择第二种形式。理由是 **MATLAB** 可以产生我们所输入的符号方程的图象。此时要使用 *ezplot* 命令。我们看看下面的例子。

首先我们创建表示方程的字串：

```
>> d = 'x^2 - 6*x - 12';
```

现在调用 *ezplot*：

```
>> ezplot(d)
```

MATLAB 的响应就是图 5-1。显示出来的字串有：

- *ezplot* 已经在图象顶端显示出标题，不需要我们做任何的额外工作。
- 图象的 *x* 轴标签。



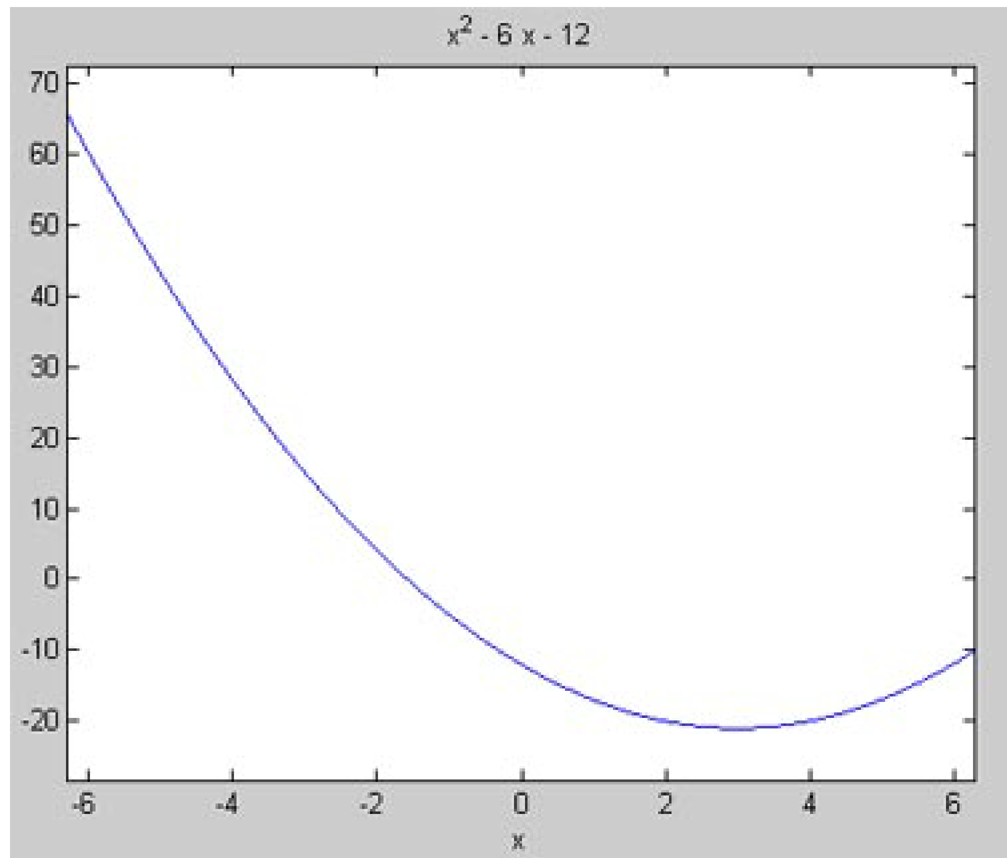


图 5-1 使用 *ezplot* 产生的符号方程图象

函数还自动选择了图象的区间范围。

当然，如我们对它所选择的区间不满意，则可以使用下面的语法指定我们所要的范围：

```
ezplot(f, [x1 , x2])
```

它绘制了 f 在 $x_1 < x < x_2$ 区间内的图象。

回头再看看刚才的例子，假设我们要绘制 $-2 < x < 8$ 内的图象。我们可以使用下面的命令：

```
>> d = 'x^2 - 6*x - 12';
>> ezplot(d, [-2, 8])
```

这次产生的图象如图 5-2。

在我们进一步深入学习之前，我们回到“=0”的事况。假设我们试图绘制：

```
>> ezplot('x+3=0')
```

MATLAB 完全不喜欢它，输出一大片完全没有意义的错误信息：

```
??? Error using ==> inlineeval
Error in inline expression ==> x+3=0
??? Error: The expression to the left of the equals sign is not a valid target for an assignment.

Error in ==> inline.feval at 34
    INLINE_OUT_ = inlineeval(INLINE_INPUTS_, INLINE_OBJ_.inputExpr, INLINE_OBJ_.expr);

Error in ==> specgraph\private\ezplotfeval at 54
    z = feval(f,x(1));

Error in ==> ezplot>ezplot1 at 448
[y,f,loopflag] = ezplotfeval(f,x);
```

```
Error in ==> ezplot at 148
    [hp,cax] = ezplot1(cax,f{1},vars,labels,args{:});
```

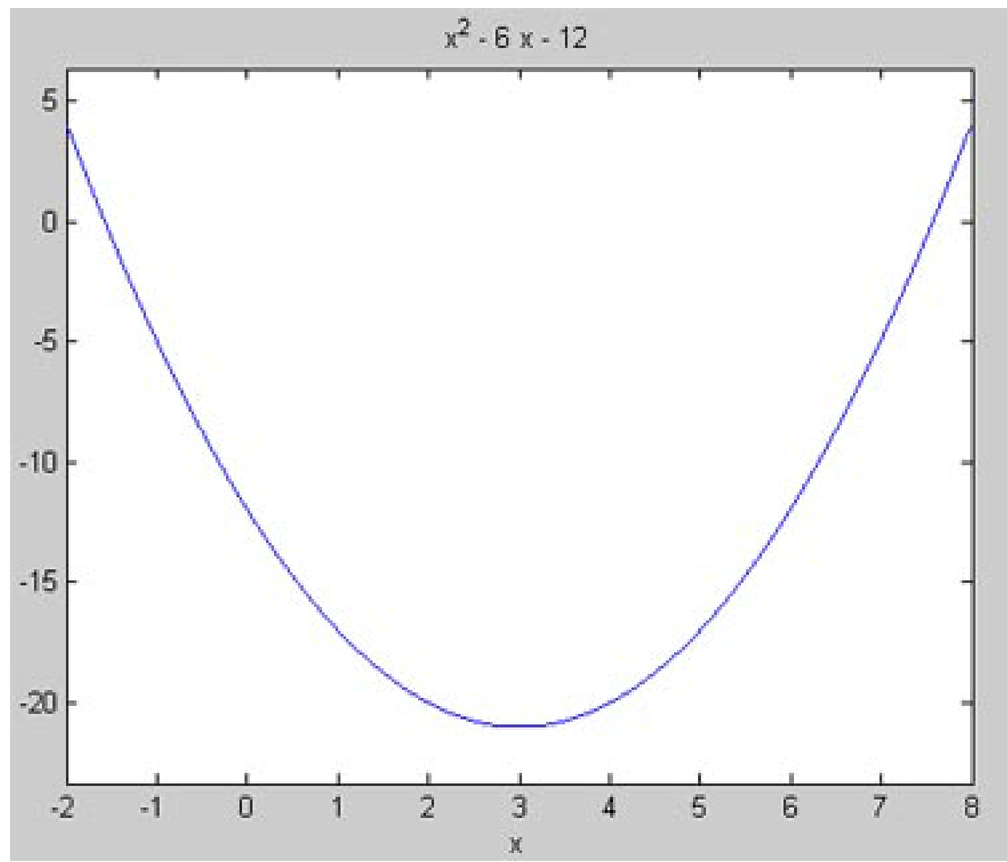


图 5-2 使用指定区间的 *ezplot* 绘制的图象

现在，如果我们把输入改成：

```
>> ezplot('x+3')
```

它会绘制出一条直线，如图 5-3。

刚才我们提到了可以给 *ezplot* 指定图象函数的定义域，自然而然，它也允许我们指定它的值域。假设我们要绘制下面的图象：

```
x + 3 = 0
-4 < x < 4, -2 < y < 2
```

我们输入如下：

```
>> ezplot('x+3', [-4, 4, -2, 2])
```

产生的图象如图 5-4。因此，要绘制 $x_1 < x < x_2$ 和 $y_1 < y < y_2$ 的图象，要在调用 *ezplot* 命令时包含 $[x_1, x_2, y_1, y_1]$。

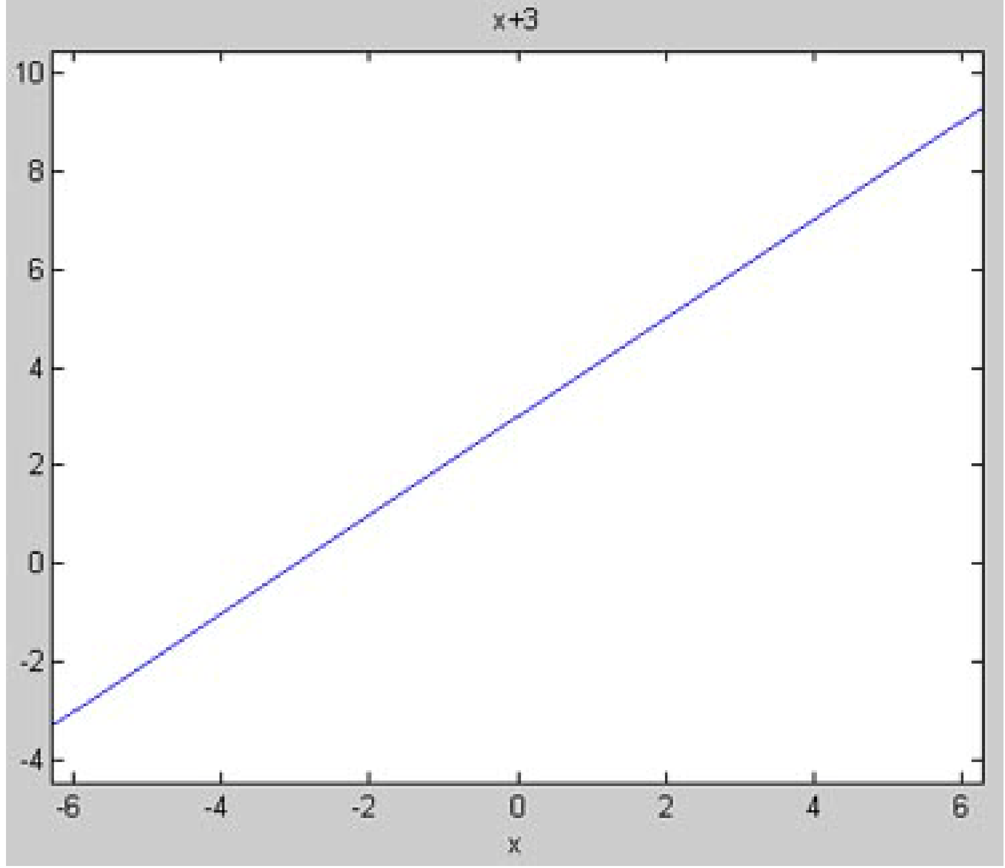


图 5-3 使用 ezplot('x + 3')正常工作，但 ezplot('x + 3 = 0')会出错

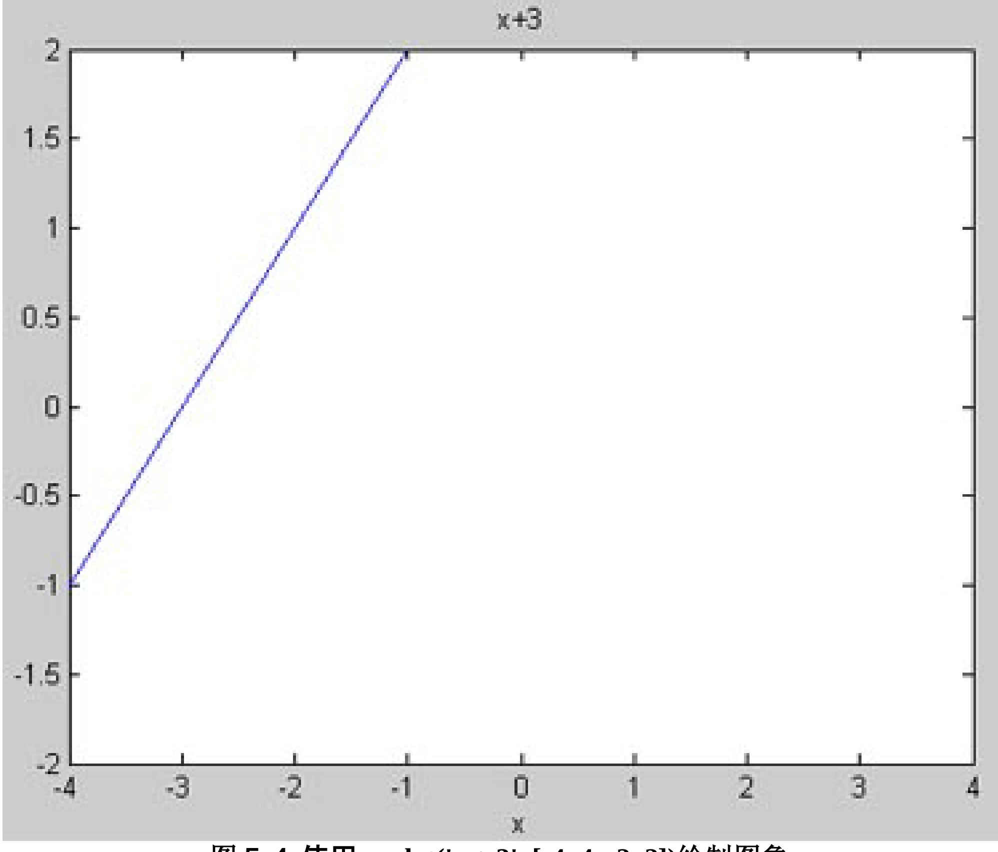


图 5-4 使用 ezplot('x + 3', [-4, 4, -2, 2])绘制图象

例 5-1

求函数$x^2 + x - \sqrt{2} = 0$ 的根并绘制该图象、确定根的数值。

解 5-1

首先我们创建一个字串表示该方程。注意你**可以**在方程中预定义 **MATLAB** 表达式。因此如下输入公式是完全可行的：

```
>> eq = 'x^2 + x - sqrt(2)';
```

如果你喜欢，你也可以写成：

```
>> eq = 'x^2 + x - 2^(1/2)';
```

下一步我们调用 *solve* 求方程的根：

```
>> s = solve(eq)
s =
-1/2+1/2*(1+4*2^(1/2))^(1/2)
-1/2-1/2*(1+4*2^(1/2))^(1/2)
```

要确定根的数值，我们需要把它们从数组中提取出来，然后把它们转换成 *double* 类型。只需简单地把它们传递给 *double(.)*命令即可。例如，我们要得到第一个根的值，写成：

```
>> x = double(s(1))
x =
    0.7900
```

而第二个根则是：

```
>> y = double(s(2))
y =
   -1.7900
```

要绘制函数的图象，我们调用 *ezplot*：

```
>> ezplot(eq)
```

结果如图 5-5 所示。

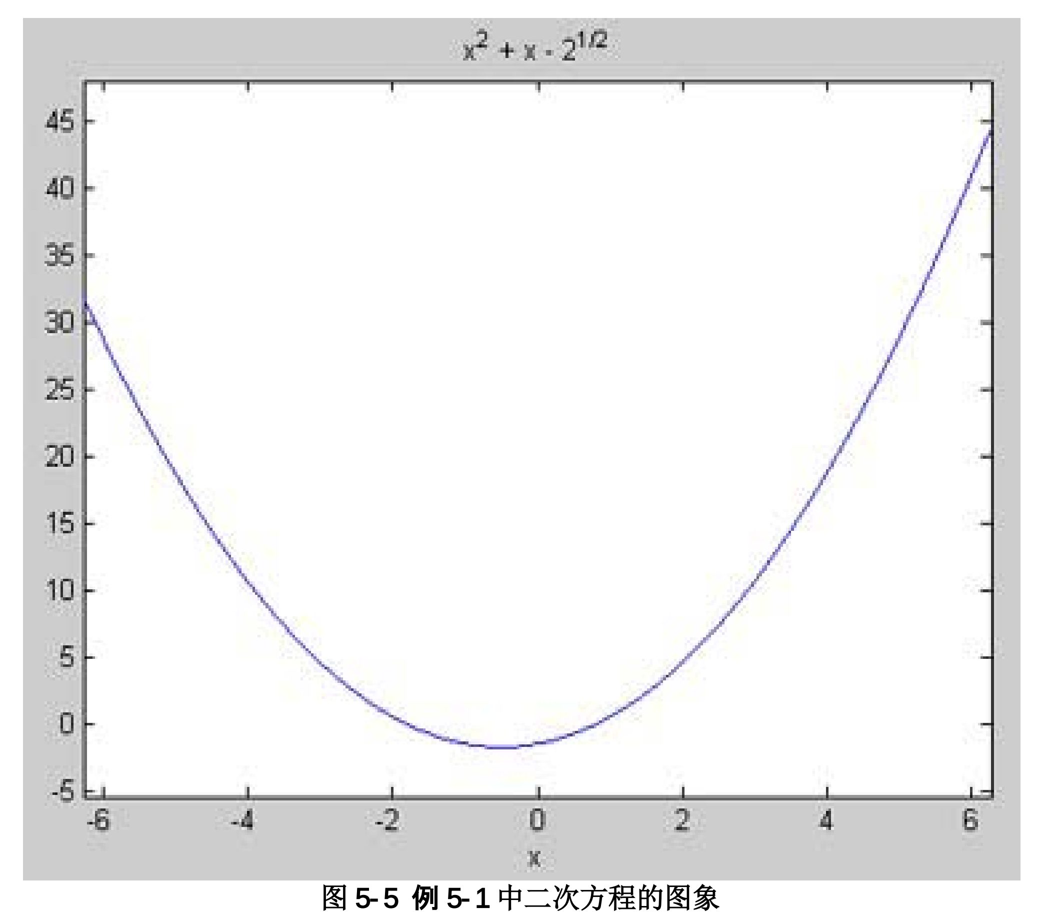


图 5-5 例 5-1 中二次方程的图象

高阶方程求解

当然你可以使用 **MATLAB** 解更高阶数的方程。我们试一个三次方的方程。假设我们有：

$$(x + 1)^2 (x-2) = 0$$

求解这样的方程并没有比前面的难。我们可以求得方程的根：

```
>> solve(eq)
ans =
 2
 -1
 -1
```

例 5-2

求四次方程的根：

$$x^4 - 5x^3 + 4x^2 - 5x + 6 = 0$$

并绘制该函数在 $-10 < x < 10$ 范围内的图象。

解 5-2

首先我们创建一个表示该方程的字串：

```
>> eq1 = 'x^4 - 5*x^3 + 4*x^2 - 5*x + 6';
```

然后我们调用 *solve* 求根：

```
>> s = solve(eq1);
```

现在我们定义一个变量从 s 中提取根。如果你把它们用符号列出来，你会得到一大堆。下面我们写出第一个根的一部分：

```
>> a = s(1)
a =
5/4+1/12*3^(1/2)*((43*(8900+12*549093^(1/2))^(1/3)+2*(8900+12*549093^(1/2))^(2/3)+104)...
```

实际操作一下你会看到这一行会很长，因此，我们使用 double 得到它的数值结果：

```
>> double(s(1))
ans =
    4.2588
```

现在的结果好多了。我们获取其它根，由于这是一个四次方程，它有四个根：

```
>> double(s(2))
ans =
    1.1164
>> double(s(3))
ans =
  -0.1876 + 1.1076i
>> double(s(4))
ans =
  -0.1876 - 1.1076i
```

注意其中有两个根是复数。现在我们在限定的定义域内绘制函数的图象：

```
>> ezplot(eq1,[-10 10])
```

结果如图 5-6。

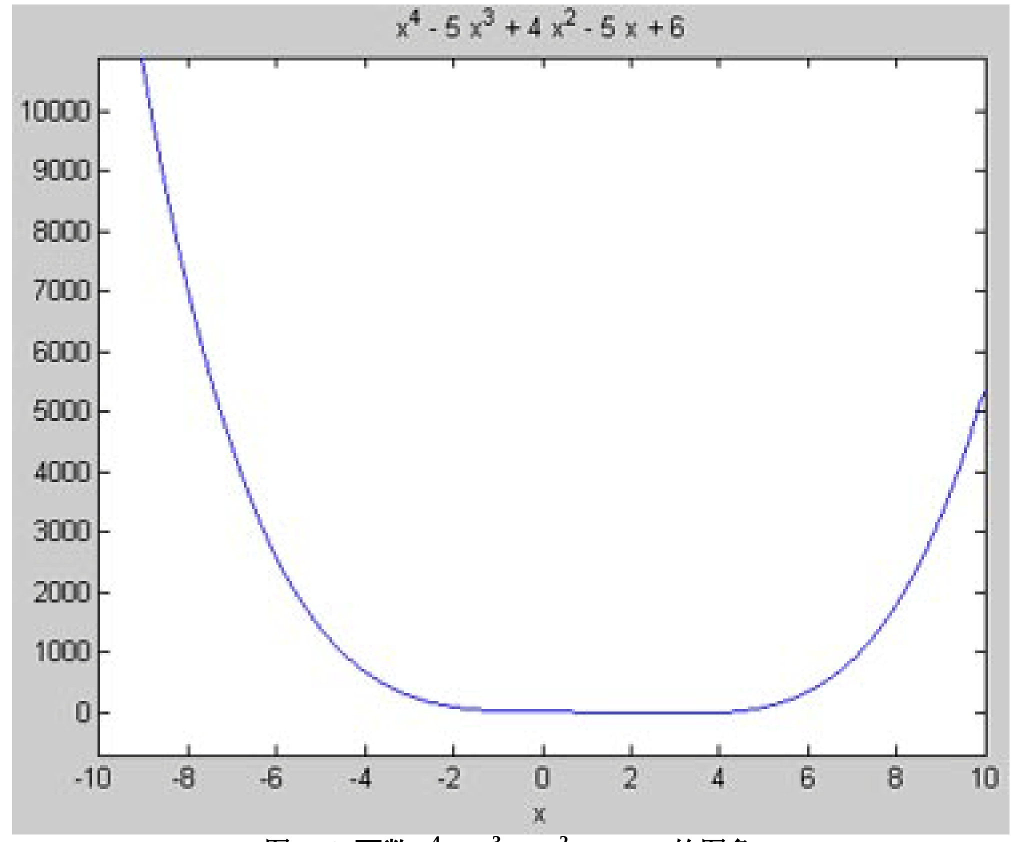


图 5-6 函数 $x^4 - 5x^3 + 4x^2 - 5x + 6$ 的图象



例 5-3

求方程 $x^3 + 3x^2 - 2x - 6$ 的根，绘制-8< x <8，-8< y <8 范围内的图象并打开网格线

解 5-3

我们再次按照一般的步骤来处理。首先定义一个字串表示该函数：

```
>> f = 'x^3 + 3*x^2 - 2*x - 6';
```

调用 *solve*，求得根-3 和$\pm\sqrt{2}$ ：

```
>> s = solve(f)
s =
        -3
  2^(1/2)
 -2^(1/2)
```

现在我们调用 *ezplot* 来产生函数的图象，并指范围-8 < x < 8，-8 < y < 8，添加 *grid on* 命令打开网格线：

```
>> ezplot(f, [-8, 8, -8, 8]), grid on
```

结果如图 5-7 所示。

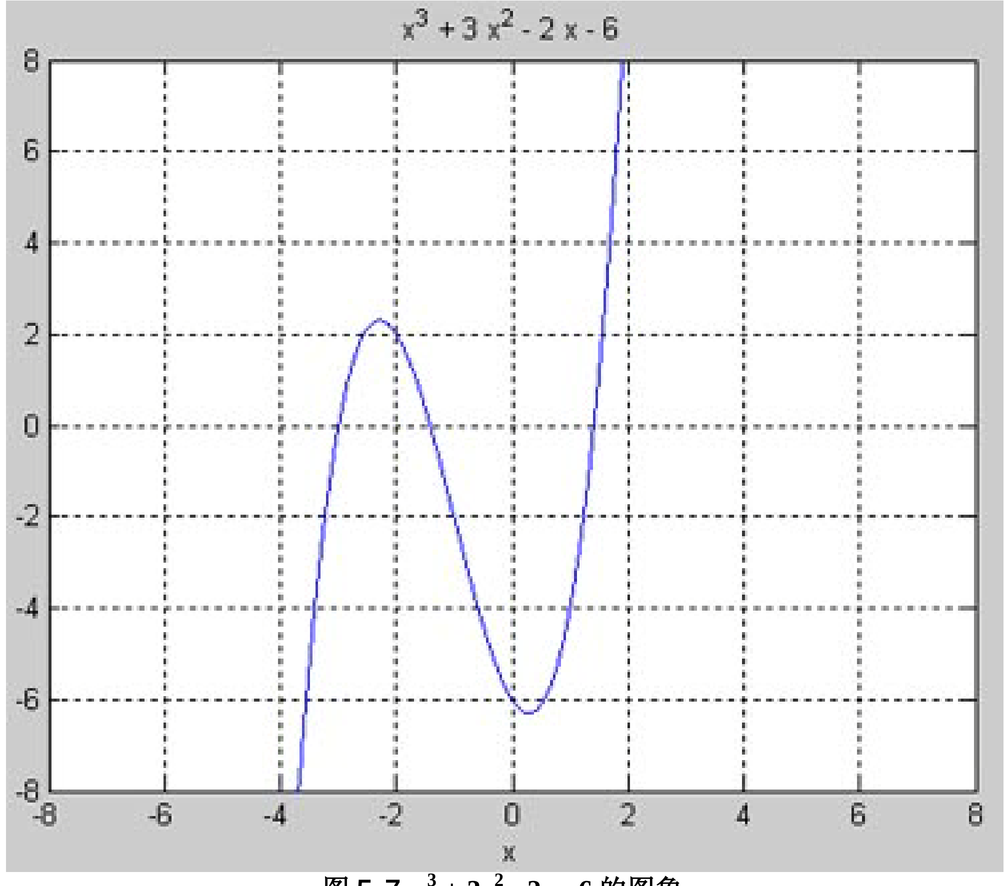


图 5-7 $x^3 + 3x^2 - 2x - 6$ 的图象

解方程组

虽然已经体会到了 *solve* 很好用，不过它实际上比我们目前所看到的更好用。*solve* 还可以用来解方程组。要知道怎么求输方程组，最好的方法就是再举另一个简单的例子。假设你有下面的方程组：

$$5x + 4y = 3$$
$$x - 6y = 2$$

要使用 *solve* 解出 x 和 y，我们要在调用它的时候传递两个参数给它——每个字串表示一个方程。在本例中我们输入：

```
>> s = solve('5*x + 4*y = 3','x - 6*y = 2');
```

注意每个方程都放在单引号内，并使用逗号“,”把方程隔开。如下，使用点句号“.”我们就可以取得相应的 x 和 y 值。首先我们得到 x：

```
>> x = s.x
x =
13/17
```

MATLAB 以分数形式给我们输出结果，我们还可以把它写到作业本上——教授是不会怀疑的。下一步我们取得 y：

```
>> y = s.y
y =
-7/34
```

这种方法还可以用来解更大的线性方程组。考虑下面的方程组：

$$w + x + 4y + 3z = 5$$
$$2w + 3x + y - 2z = 1$$
$$w + 2x - 5y + 4z = 3$$
$$w - 3z = 9$$

我们可以以一组字串的形式输入方程组：

```
>> eq1 = 'w + x + 4*y + 3*z = 5';
>> eq2 = '2*w +3*x+y - 2*z = 1';
>> eq3 = 'w + 2*x - 5*y + 4*z = 3';
>> eq4 = 'w - 3*z = 9';
```

现在我们调用 *solve*，把方程组用逗号“,”分开传递给它：

```
>> s = solve(eq1,eq2,eq3,eq4);
```

然后我们就可以提取方程组中各个变量了：

```
>> w = s.w
w =
1404/127

>> x = s.x
x =
-818/127

>> y = s.y
```



```
y =
-53/127

>> z = s.z
z =
87/127
```

方程展开与合并

初中时我们学过如何展开方程，例如：

$$(x + 2)(x - 3) = x^2 - x - 6$$

我们可以使用 **MATLAB** 来完成这一类任务，只需要调用 *expand* 命令。使用 *expand* 命令相对简单，例如：

```
>> syms x
>> expand((x - 1)*(x + 4))
```

当执行命令后，我们得到：

```
ans =
x^2+3*x-4
```

这个 *expand* 函数还可以应用到其它形式，例如，我们可以应用三角函数，产生著名的三角恒等式：

```
>> syms y
>> expand(cos(x+y))
```

输出：

```
ans =
cos(x)*cos(y)-sin(x)*sin(y)
```

或者：

```
>> expand(sin(x-y))
ans =
sin(x)*cos(y)-cos(x)*sin(y)
```

要使用含有多个符号的函数，你必须告诉 **MATLAB** 有哪些符号变量。例如，如果我们输入：

```
>> expand((y-2)*(y+8))
```

MATLAB 返回：

```
??? Undefined function or variable 'y'.
```

要避免错误提示，首先必须输入：

```
>> syms y
```

然后我们才得到：

```
>> expand((y-2)*(y+8))
ans =
y^2+6*y-16
```

MATLAB 也让我们在其它方面使用——合并和化简方程。首先我们看看如何使用 *collect* 命令。你可能需要它的一个地方是分配多项式。考虑下面的方程：

$$x(x^2 - 2) = x^3 - 2x$$

在 **MATLAB** 我们只需输入下面命令即可做到：

```
>> syms x;
>> collect(x*(x^2 - 2))
ans =
x^3-2*x
```

另一个例子（带有一个新的变量符号 *t*）

```
>> syms t
>> collect((t+3)*sin(t))
ans =
sin(t)*t+3*sin(t)
```

对于处理符号，我们另一个代数任务是因式分解。使用下面的例子做为演示：

$$x^2 - y^2 = (x + y)(x - y)$$

使用 **MATLAB**，我们输入：

```
>> syms x;syms y;
>> factor(x^2 - y^2)
ans =
 (x-y)*(x+y)
```

我们还可以使用一个命令同进对多个方程式进行因式分解：

```
>> factor([x^2-y^2, x^3+y^3])
ans =
       [(x-y)*(x+y), (x+y)*(x^2-x*y+y^2)]
```

最后，我们使用 *simplify* 命令。这个命令可以进行多项式相除。例如，我们要演示 $(x^2 + 9)(x^2 - 9) = x^4 - 81$，如下：

```
>> syms x;
>> simplify((x^4-81)/(x^2-9))
ans =
x^2+9
```

下面是另一个例子。考虑到：

$$e^{2\log 3x} = e^{\log 9x^2} = 9x^2$$

我们可以使用 **MATLAB** 命令把结果给求出来：

```
>> syms x;
>> simplify(exp(2*log(3*x)))
ans =
9*x^2
```



simplify 命令在求解三角恒等式时同样有用。例如：

```
>> syms x;
>> simplify(cos(x)^2-sin(x)^2)
ans =
2*cos(x)^2-1
>> simplify(cos(x)^2+sin(x)^2)
ans =
1
```

使用指数和对数函数求解方程

到目前为止我们使用的例子都是考虑多项式。符号求解器还允许使用指数和对数。首先我们考虑带有对数的方程。

例 5-4

求出满足下式的 x 值：

$$log_{10}(x)-log_{10}(x-3)=1$$

解 5-4

在 **MATLAB** 中，以 10 为底的对数可以用 *log*10 计算或表示。因此我们输入如下内容：

```
>> eq = 'log10(x) - log10(x - 3) = 1';
```

然后调用 *solve*：

```
>> s = solve(eq);
```

在本例中，**MATLAB** 只返回一个解：

```
>> s(1)
ans =
10/3
```

现在我们再看看指数为变量的一些方程。假设要我们解下面的方程组：

$$y = 3^{2x}$$
$$y = 5^{x} + 1$$

我们可以使用 *solve* 求得方程的解：

```
>> s = solve('y = 3^2*x','y = 5^x+1')
```

MATLAB 返回：

```
s =
    x: [2x1 sym]
    y: [2x1 sym]
```

看来方程的解有两个变值。我们可以把 *x* 值拿出来，输入如下：

```
>> s.x(1)
ans =
1/9*exp(-lambertw(-1/9*log(5)*5^(1/9))+1/9*log(5))+1/9
```

```
>> s.x(2)
ans =
1/9*exp(-lambertw(-1,-1/9*log(5)*5^(1/9))+1/9*log(5))+1/9
```

这些结果可能对一般的人来说没有什么用。因此我们把它们转换成数值：

```
>> a = double(s.x(1))
a =
    0.2876
>> b = double(s.x(2))
b =
    1.6214
```

我们还可以提取 *y* 值：

```
>> c = double(s.y(1))
c =
    2.5887
>> c = double(s.y(2))
c =
   14.5924
```

结果好理解多了！我们检验一下，看看 *s.x*(1)是否与 *s.y*(1)同时满足第一个方程，同样也检验第二组数值。我们得到：

```
>> 3^2*a
ans =
    2.5887
```

到目前为止表现良好，这与 *s.y*(1)相一致。现在：

```
>> 5^b+1
ans =
   14.5924
```

因此，*s.y*(2)和 *s.x*(2)也相一致，我们得解。
我们也可以输入和求解含有指数的方程。例如：

```
>> eq = 'exp(x)+ x';
>> s = solve(eq)
s =
-lambertw(1)
```

如果你对兰伯特(*Lambert*)的 *W* 方程不熟悉，我们可以计算它的数值：

```
>> double(s)
ans =
   -0.5671
```

我们使用 *ezplot* 把函数绘制出来：：

```
>> ezplot(eq)
```

结果如图 5-8 所示：

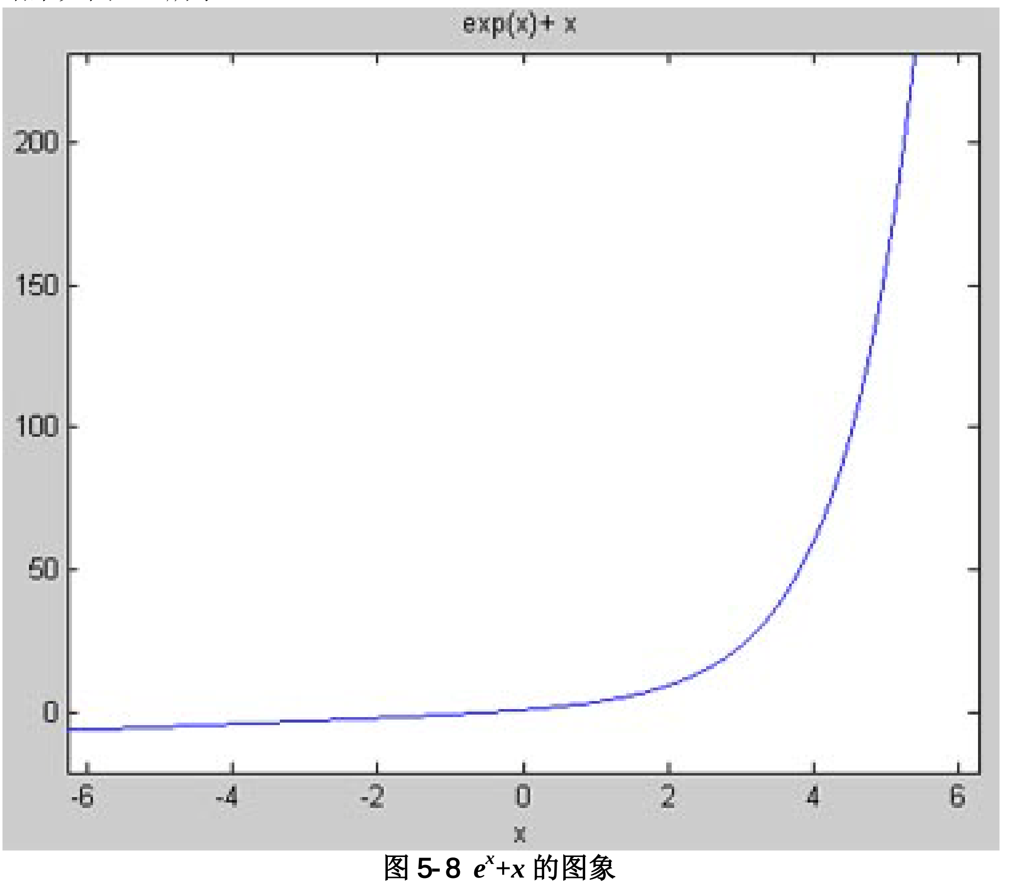


图 5-8 e^x+x 的图象

函数的级数表示

在本章的最后，我们学习如何使用 **MATLAB** 得到用符号表示的函数的级数表达式。*taylor* 函数返回某个函数的泰勒(*Taylor*)级数展开式。最简单的方法是直接调用 *taylor* 函数。例如要得到 sin(*x*)函数的泰勒级数展开式：

```
>> syms x
>> s = taylor(sin(x))
s =
x-1/6*x^3+1/120*x^5
```

MATLAB 返回了展开式的前三项。事实上 **MATLAB** 返回的是：

$$\sum_{n=0}^{5}\frac{f^{(n)}(0)}{n!}x^n$$

在上面的 sin 例子中，返回的是非零项。下面我们绘制它的图象：

```
>> ezplot(s)
```

图象如图 5-9 所示。

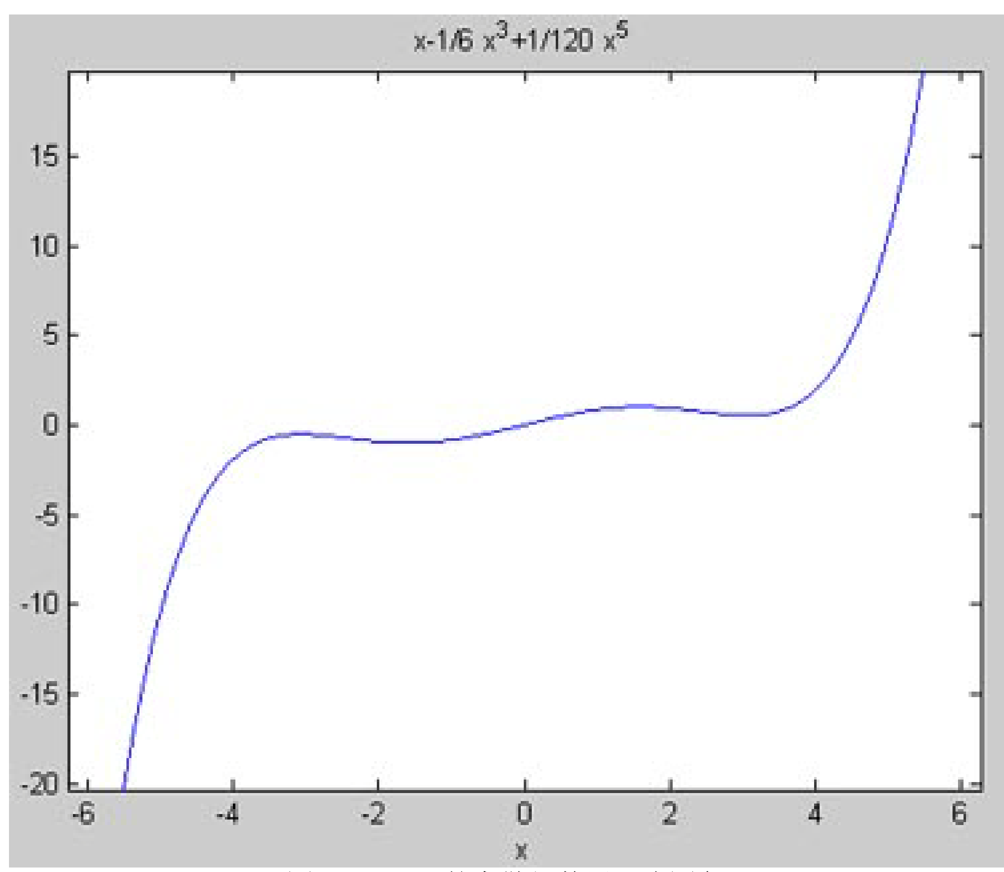


图 5-9 sin(*x*)的泰勒级数展开式图象

对于小范围，这个表达式可能会比较精确，不过在大范围的定义区间内来看，它确实不怎么像 sin(*x*)。要使 **MATLAB** 返回更多项，假设我们要得到 *m* 项展开式：

$$\sum_{n=0}^{m}\frac{f^{(n)}(0)}{n!}x^n$$

我们写成 *taylor(f, m)*。对于当前的例子，我们试一下前 20 项：

```
>> syms x;
>> s = taylor(sin(x),20)
s =
x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9-1/39916800*x^11+
1/6227020800*x^13-1/1307674368000*x^15+1/355687428096000*x^17-
1/121645100408832000*x^19
```

这一次表达式要精确得多了，相应的图象如图 5-10 所示：

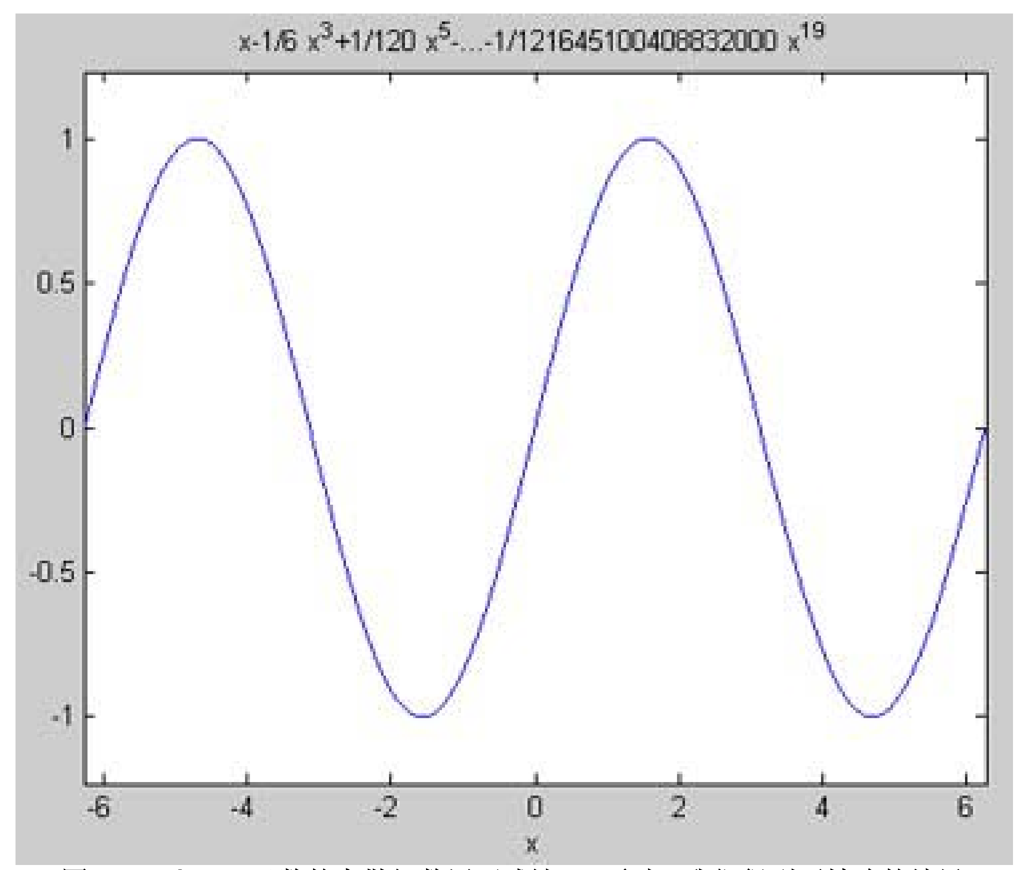


图 5-10 当 sin 函数的泰勒级数展开式达 20 项时，我们得到更精确的结果

习题

1. 在 **MATLAB** 中以字串形式输入$7\sqrt{2} - 5\sqrt{60} + 5\sqrt{8}$，然后求它的值。
2. 使用 **MATLAB** 解 $3x^2 + 2x = 7$。
3. 求$x^2 + \sqrt{5}x - \pi = 0$ 中的 x 值。
4. 求$\sqrt{2x - 4} = 1$ 的解，并绘制该函数在 $2 < x < 4,\ 0 < y < 1$ 内的图象。
5. 使用 solve 求 $2t^3 - t^2 + 4t - 6 = 0$ 的根，然后把答案转换成数值。
6. 求下列方程组的解：

$$\begin{aligned} x - 3y - 2z &= 6 \\ 2x - 4y - 3z &= 8 \\ -3x + 6y + 8z &= -5 \end{aligned}$$

7. 方程有 $e^x - x^2 = 0$ 实数根吗？
8. 使用 **MATLAB** 化简 $\tan^2 x - \sec^2 x$。
9. 求 $\tan(x)$的泰勒展开式的第十项。
10. 求$\frac{4}{5 - \cos x}$ 的泰勒级数展开式的第十项，结果与图 5-11 一致吗？

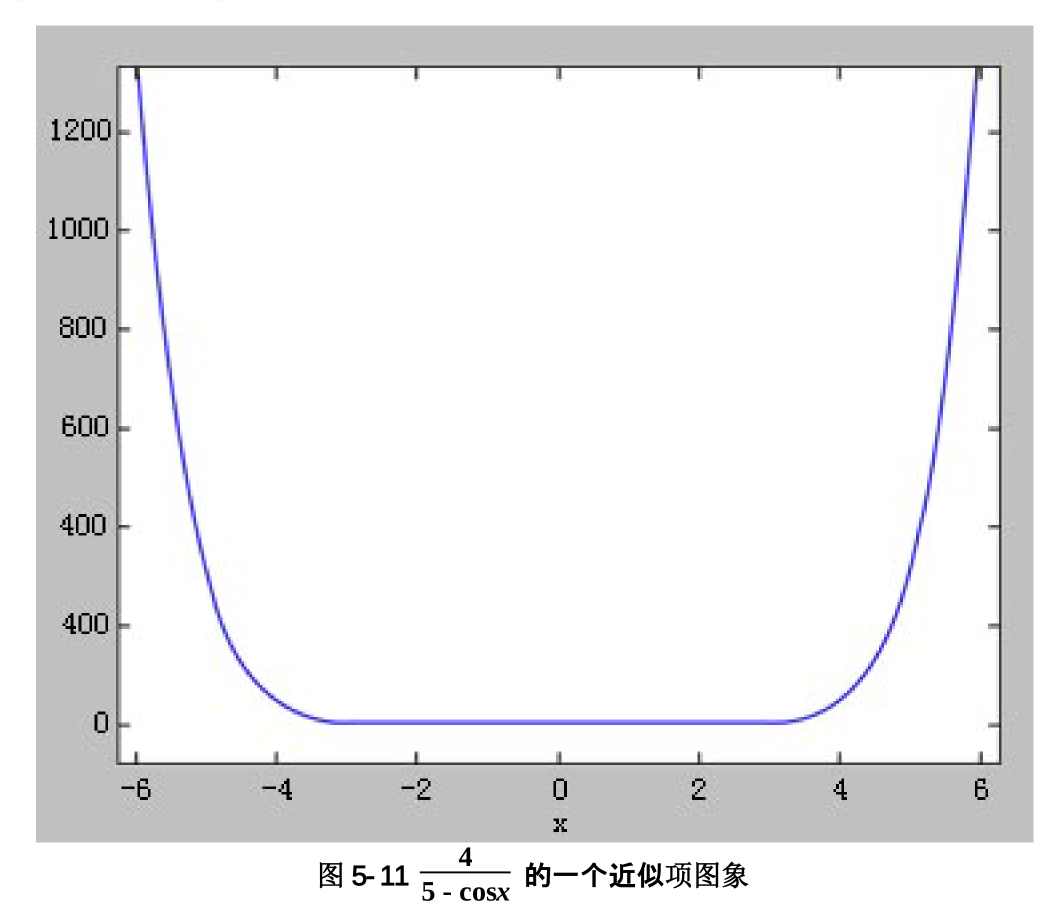


图 5-11 $\frac{4}{5-\cos x}$ 的一个近似项图象

【参考答案在第 229 页】



第六章

基本符号演算和微分方程

本章我们将学习如何使用 **MATLAB** 进行符号演算。特别地，我们从学习极限和导数开始，然后解微分方程。下一章我们再学习积分。

极限计算

MATLAB 可以使用 *limit* 命令计算极限，最基本的使用方法就是输入你要计算的表达式。**MATLAB** 将会帮你找出独立变量趋于零时的极限。例如，如果你输入函数 $f(x)$，**MATLAB** 就会求出 $\lim_{x\to 0} f(x)$。下面是一个例子：

```
>> syms x
>> limit((x^3 + 1)/(x^4 + 2))
ans =
1/2
```

记住，*limit* 命令是符号计算方面的内容，因此确保使用 *syms* 命令告诉 **MATLAB** 你使用的是哪个符号变量。

要计算 $\lim_{x\to a} f(x)$，*limit* 命令使用的语法是 *limit*$(f,\ a)$。例如：

```
>> limit(x + 5,3)
ans =
8
```

例 6-1

设 $f(x) = \frac{2x+1}{x-2}$ 和 $g(x) = x^2 + 1$。计算这两个函数 $x\to 3$ 时的极限，并在 **MATLAB** 中使用它们验证极限的基本性质。

解 6-1

首先我们告知 **MATLAB** 我们使用的符号变量是什么，然后定义函数：

```
>> syms x
>> f = (2*x + 1)/(x-2);
>> g = x^2 + 1;
```

现在我们计算每个函数的极限，并把结果保存起来以便后面使用：

```
>> F1 = limit(f,3)
F1 =
7
>> F2 = limit(g,3)
F2 =
10
```

我们要验证的第一个极限性质是：

$$\lim_{x\to a}(f(x) - g(x)) = \lim_{x\to a} f(x) - \lim_{x\to a} g(x)$$

从目前我们计算的结果可以看到：

$$\lim_{x\to 3} f(x) - \lim_{x\to 3} g(x) = 7 + 10 = 17$$

现在我们通过计算左边的值来验证它们的关系：

```
>> limit(f+g,3)
ans =
17
```

下一步我们验证：

$$\lim_{x\to a} k\ f(x) = k \lim_{x\to a} f(x)$$

其中 k 是任意常数。让 k=3 因此我们应该求得：

$$\lim_{x\to a} k\ f(x) = 3 \lim_{x\to 3} f(x) = (3)(7) = 21$$

```
>> k=3;
>> limit(k*f,3)
ans =
21
```

现在我们验证两个函数乘积的极限等于两个函数各自极限的乘积，也就是：

$$\lim_{x\to a}(f(x)\ g(x)) = \lim_{x\to a} f(x) \lim_{x\to a} g(x)$$

极限的乘积是：

```
>> F1*F2
ans =
70
```

我们求得的俩函数乘积的极限是：

```
>> limit(f*g,3)
ans =
70
```

最后，我们验证：

$$\lim_{x\to a} f(x)^{g(x)} = \lim_{x\to a} f(x)^{\lim_{x\to a} g(x)}$$

我们在 **MATLAB** 创建 $f(x)^{g(x)}$：

```
>> h = f^g
h =
 ((2*x+1)/(x-2))^(x^2+1)
```

计算它的极限：

```
>> limit(h,3)
ans =
282475249
```

检查与右边的关系，我们发现它们是相等的：

```
>> A = F1^F2
A =
282475249
```

题外话，我们可以在 **MATLAB** 中调用 `isequal` 命令检查两个量是否相等，如果两个量不相等，`isequal` 返回 0。回头看看我们前面定义 k=3 的例子，如果我们把它跟 *A=F1^F2* 进行比较，**MATLAB** 返回：

```
>> isequal(A,k)
ans =
     0
```

另一方面：

```
>> isequal(A,limit(h,3))
ans =
     1
```

计算$\lim_{x\to\infty} f(x)$

我们使用下面的语法可以计算$\lim_{x\to\infty} f(x)$形式的极限：

```
limit(f,inf)
```

我们使用 **MATLAB** 来显示$\lim_{x\to\infty}(\sqrt{x^2+x}-x)=\frac{1}{2}$：

```
>> limit(sqrt(x^2+x)-x,inf)
ans =
1/2
```

我们同样能计算$\lim_{x\to-\infty} f(x)$。例如：

```
>> limit((5*x^3 + 2*x)/(x^10 + x + 7),-inf)
ans =
0
```

MATLAB 同样能够告诉我们极限的结果是∞。例如，我们验证$\lim_{x\to 0}\frac{1}{|x|}=\infty$：

```
>> limit(1/abs(x))
ans =
Inf
```

左极限和右极限

如果存在间断点，那以函数在该点的极限就不存在。为了处理函数在 $x=a$ 处不连续的情况，我们定义了函数左极限和右极限的记号。左极限定义为 x 从左边趋近于 a 时的极限，即在 $x<a$ 的情况下 $x\to a$。在符号演算中我们写成：

$$\lim_{x\to a^-} f(x)$$

右极限是 x 从右边趋近 a 时的极限，即在 $x>a$ 的情况下 $x\to a$。我们用来表示右极限的记号是：

$$\lim_{x\to a^+} f(x)$$

如果这些极限存在且相等，那么$\lim_{x\to a} f(x)$就存在。在 **MATLAB** 中，我们通过给 *limit* 命令传递 “*left*”和“*right*”字串作为最后一个参数来计算函数的左极限和右极限。我们还必须告诉 **MATLAB** 用来计算极限的变量。让我们用一个例子来说明：

例 6-2

指出$\lim_{x\to 3}\frac{x-3}{|x-3|}$不存在。

解 6-2

首先在 **MATLAB** 中定义函数：

```
>> syms x;
>> f = (x - 3)/abs(x - 3);
```

如果我们绘制函数的图象，$x=3$ 这一点不连续是很明显的，如图 6-1。注意为了要显示图象的间断点，我们给 **MATLAB** 指定了定义域：

```
>> ezplot(f,[-1,5])
```

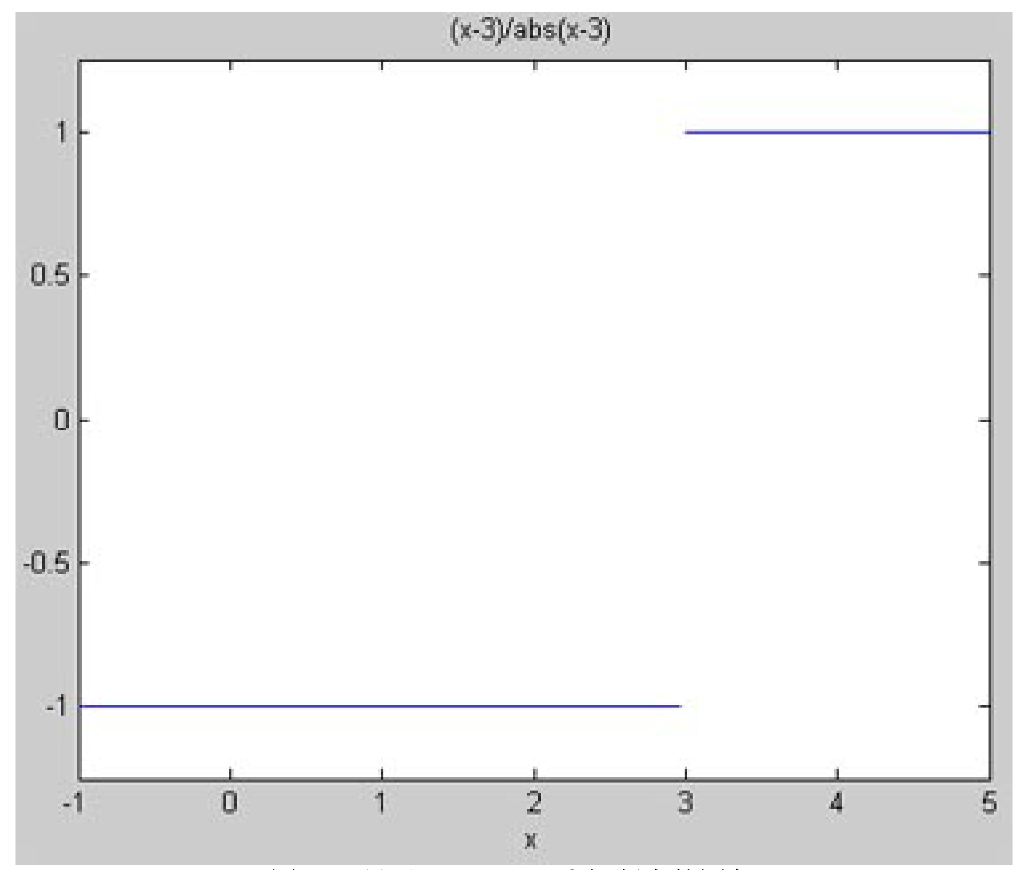


图 6-1 显示$(x - 3)/|x - 3|$中间断点的图象

现在我们计算左极限。要计算它，我们必须给函数传递用来计算极限的变量和“*left*”字串，并用逗号分隔开：

```
>> a = limit(f,x,3,'left')
a =
-1
```

现在我们取右极限：

```
>> b = limit(f,x,3,'right')
b =
1
```

由于这两个结果不相等，这就说明了$\lim_{x\to 3}\frac{x-3}{|x-3|}$的极限不存在。

获得渐近线

limit 命令可以用来获得函数的渐近线。让我们看看如何获得渐近线并绘制图象，把函数和它的渐近线都显示出来。

我们用下面的函数作为例子：

$$y = \frac{1}{x(x - 1)}$$

我们选用这个例子是因为它的渐近线非常清楚——函数在 $x=0$ 和 $x=1$ 变化非常快。让我们先绘制函数的图象：

```
>> syms x
>> f = 1/(x*(x-1));
>> ezplot(f)
```

结果如图 6-2 所示。可以看到，函数在两个点附近快速转向无穷大。现在我们继续找出这些点（假设我们不知道）然后把它们显示在图形上。

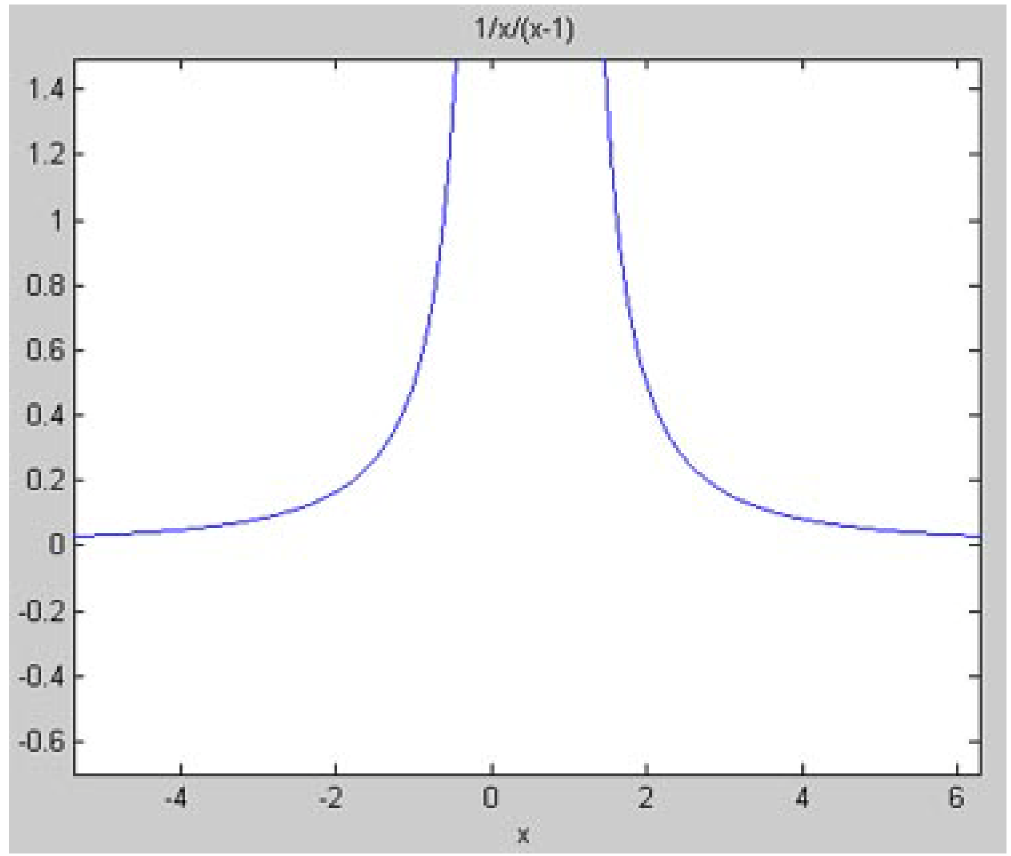


图 6-2 函数在两个点处快速变化的图象

第一步是找出函数突然增大的点。这可以通过解分母的根求得。让我们创建一个函数来表示分母，然后求它的根。回想上一章我们学习的内容，我们使用 *solve* 命令：

```
>> g = x*(x-1);
>> s = solve(g)
s =
 0
 1
```

现在手头上有根了，我们就知道渐近线在哪里了。我们把它们用虚线画出来。首先重绘函数的图象，然后使用 *hold on* 命令告诉 **MATLAB** 我们还要为图象再添加一些材料：



```
>> ezplot(1/g)
>> hold on
```

既然我们把根存在变量 s 中，记住我们访问它们是使用 *s*(1)和 *s*(2)。我们使用下面的命令即可绘制第一条渐近线：

```
>> plot(double(s(1))*[1 1], [-1 2],'--')
```

区间[-1 2]指定了我们绘制的线在 *y* 轴上的范围。尝试使用其它值，看看它是怎么工作的。现在让我们绘制第二渐近虚线：

```
>> plot(double(s(2))*[1 1], [-1 2],'--')
>> hold off
```

我们通过调用 *hold off* 命令释放 **MATLAB** 总是停留在当前的图象中。结果如图 6-3。

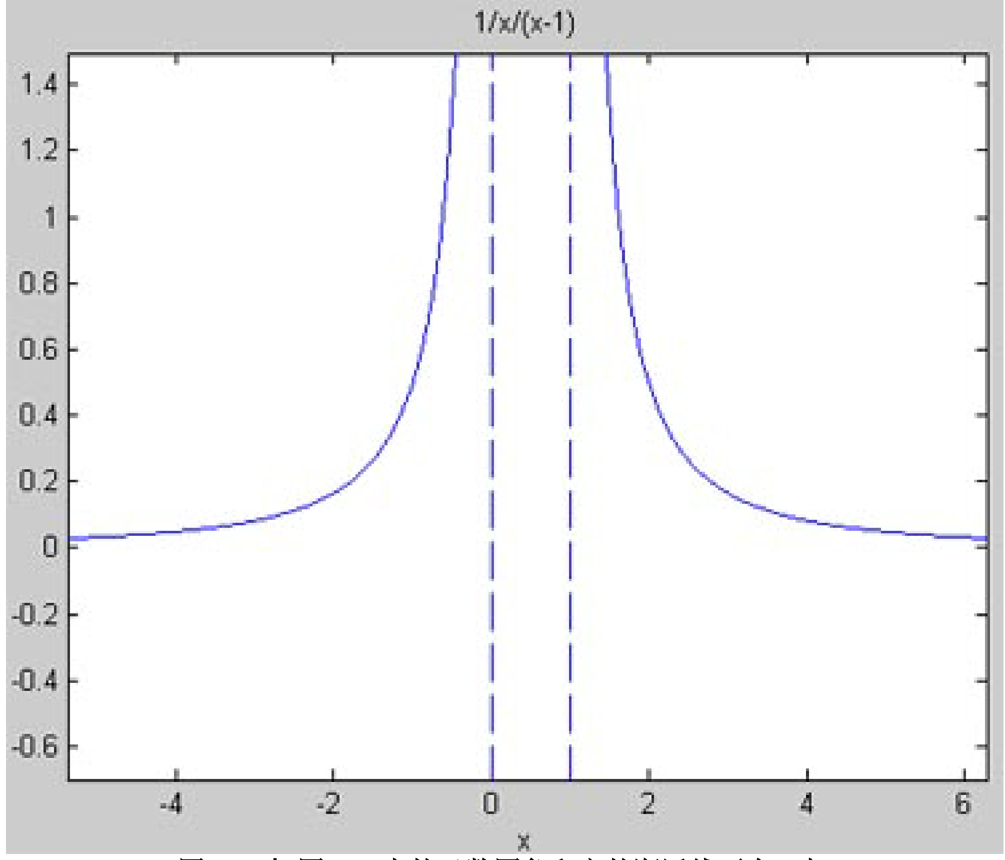


图 6-3 把图 6-2 中的函数图象和它的渐近线画在一起

导数计算

通过调用 *diff* 命令，我们可以使用 **MATLAB** 计算符号导数。只需简单地把要求导的函数传递给 *diff* 命令，如下例子所示：

```
>> syms x t
>> f = x^2;
>> g = sin(10*t);
```

```
>> diff(f)
ans =
2*x
>> diff(g)
ans =
10*cos(10*t)
```

要得到函数 *f* 的更高阶的导数，我们使用 *diff(f,n)*。让我们求 te^{-3t} 的二阶导数：

```
>> syms t
>> f = t*exp(-3*t);
>> diff(f,2)
ans =
-6*exp(-3*t)+9*t*exp(-3*t)
```

正如你所见到的，*diff* 返回求导结果——因此我们可以把结果赋给另一个变量，以便后面使用：

例 6-3

证明 $f(x) = x^2$ 满足方程 $-\frac{df}{dx} + 2x = 0$

解 6-3

首先我们做一些定义：

```
>> syms x
>> f = x^2; g = 2*x;
```

现在我们计算所需的导数：

```
>> h = diff(f);
```

最后，检验是否满足关系：

```
>> -h + g
ans =
0
```

例 6-4

$y(t) = 3\sin t + 7$ 是方程 $y'' + y = -5\cos 2t$ 的解吗？

解 6-4

定义函数：

```
>> syms t
>> y = 3*sin(t)+7*cos(5*t);
```

现在我们创建一个变量储存所需结果：

```
>> f = -5*cos(2*t);
```

输入微分方程的左边，我们创建另一个变量：

```
>> a = diff(y,2)+y;
```

我们使用 *isequal* 检查方程是否满足：

```
>> isequal(a,f)
ans =
     0
```

由于返回 0，所以 $y(t) = 3\sin t + 7$ 不是方程 $y'' + y = -5\cos 2t$ 的解。

例 6-5

求函数 $f(x) = x^3 - 3x^2 + 3x$ 在区间[0, 2]内的最小值和最大值。

解 6-5

首先我们输入函数并绘制该函数在给定区间内的图象。

```
>> syms x
>> f = x^3-3*x^2+3*x;
>> ezplot(f, [0 2])
```

图象如图 6-4 所示。

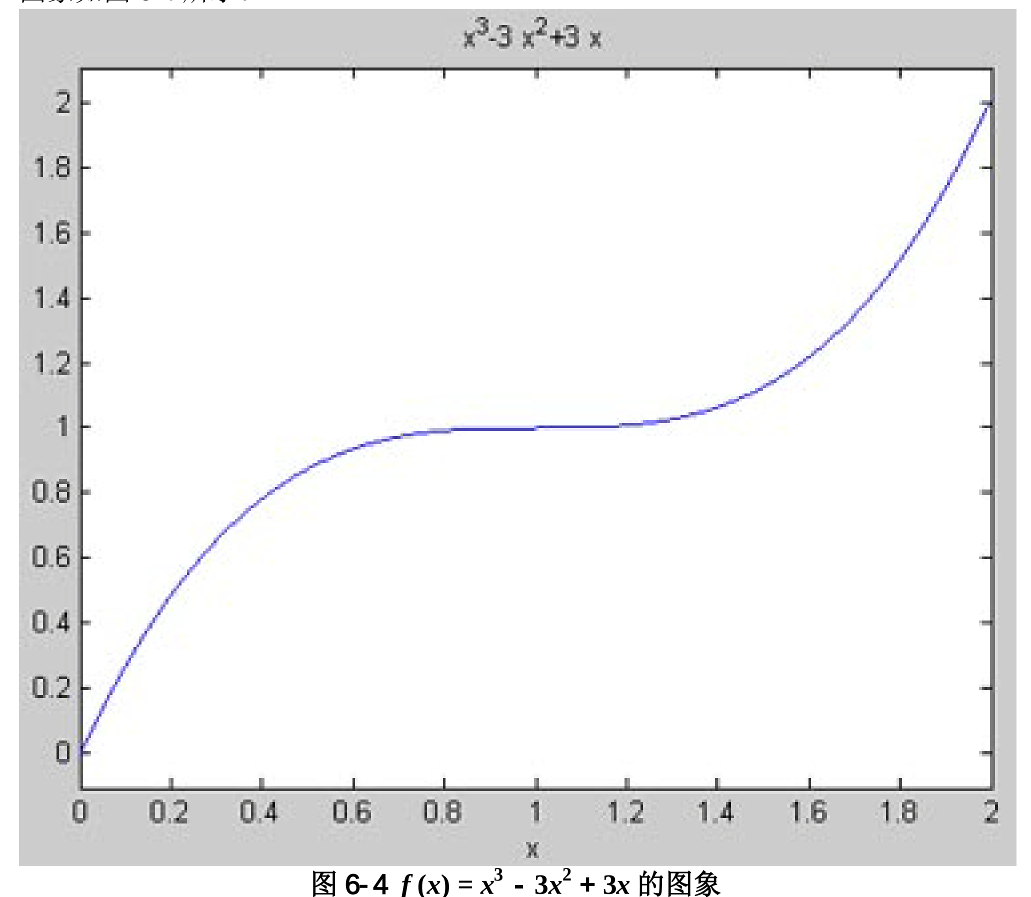


图 6-4 $f(x) = x^3 - 3x^2 + 3x$ 的图象

要求出最小值和最大值，我们求导数并找出等于零的点。导数是：

```
>> g = diff(f)
g =
3*x^2-6*x+3
```

快速插入语：我们可以使用 *pretty* 命令让表达式更好看一些：

```
>> pretty(g)

                     2
                  3 x  - 6 x + 3
```

现在稍微好看一点。回到我们的题目中，我们让导数等于零，然后求出它的根：

```
>> s = solve(g)
s =
1
1
```

虽然导数有两个根，但我们看到实际上只有一个临界点。我们可以从图象中看出最大值出现在终点，不过让我们通过计算函数在临界点 *x*=0,1,2 的值来证明它。

我们可以使用 *subs* 命令代入符号函数中的某个值，如果只有一个变量的话是相当简单的。如果想设置 *x*=*c*，那我们就调用 *subs*(*f*,*c*)。下面我们求 *x*=0,1,2 时 *f* 的值。我们可以计算这三个点的值，如下使用逗号分隔各个命令让 **MATLAB** 一次性报告这些值：

```
>> subs(f,0), subs(f,1), subs(f,2)
ans =
     0
ans =
     1
ans =
     2
```

由于 *f*(2)返回最大值，我们可以得到 *x*=2 时函数 *f* 取得最大值的结论。另外，我们还可以计算导数在这三个点的值并绘制它：

```
>> subs(g,0),subs(g,1),subs(g,2)
ans =
     3
ans =
     0
ans =
     3
```

导数的临界点在哪里呢？我们求二阶导数并让它等于零：

```
>> h = diff(g)
h =
6*x-6
>> solve(h)
ans =
1
```

再多一阶的导数是：

```
>> y = diff(h)
y =
6
```

由于 *g*''>0 我们可以下结论 *x*=1 是本区间的最小值。*g*(*x*)的图象如图 6-5

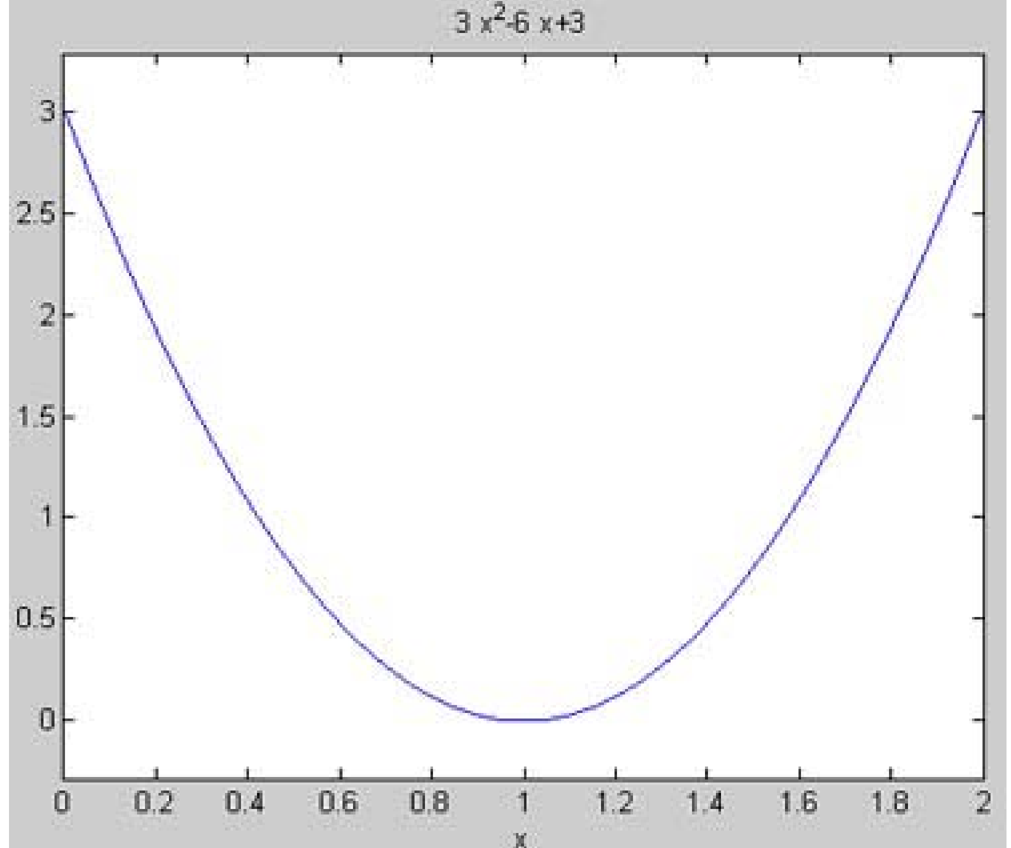


图 6-5 例 6-5 中 g(x) 的图象，显示了我们要求的最小值

例 6-6

绘制 $f(x) = x^4 - 2x^3$ 的图象并显示最大值和最小值。

解 6-6

首先我们定义函数并绘制它的图象：

```
>> clear
>> syms x
>> f = x^4-2*x^3;
>> ezplot(f,[-2 3])
>> hold on
```

现在计算一阶导数：

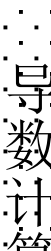
```
>> g = diff(f)
g =
4*x^3-6*x^2
```

我们把导数设为零然后求解临界值：

```
>> s = solve(g)
s =
3/2
   0
   0
```

下一步我们计算二阶导数：

```
>> h = diff(g)
```



```
h =
12*x^2-12*x
```

计算第一个临界点 x=3/2 的值，我们有：

```
>> a = subs(h,s(1))
a =
9
```

由于 f''(3/2)=9>0，所以 x=3/2 是最小值点。现在我们检查临界点的情况：

```
>> b = subs(h,s(2))
b =
0
```

在这种情况下 f''(0) = 0，这意味着无法使用二阶导数检验得到任何有用信息。从图象上可以看出这个点既不是最小值也不是最大值。现在我们修改图象让它显示最小值点。我们使用下面的命令添加 c=s(1)的点(c,f(c))：

```
>> plot(double(s(1)),double(subs(f,s(1))),'ro')
```

这一行命令在图象上(3/2, f(3/2))点画出一个红色的小圆圈。我们使用“ro”代替“o”告诉 **MATLAB** 画一个红色的小圆圈，使用“o”会画一个黑色的小圆圈。现在我们使用 *text* 命令为这个最小值点加上标签。当调用 *text* 时，你必须告诉它文本应该在哪个坐标打印出来，同时传递所要打印的文本：

```
>> text(0.8,3.2,'局部最小值')
>> hold off
```

结果如图 6-6 所示。

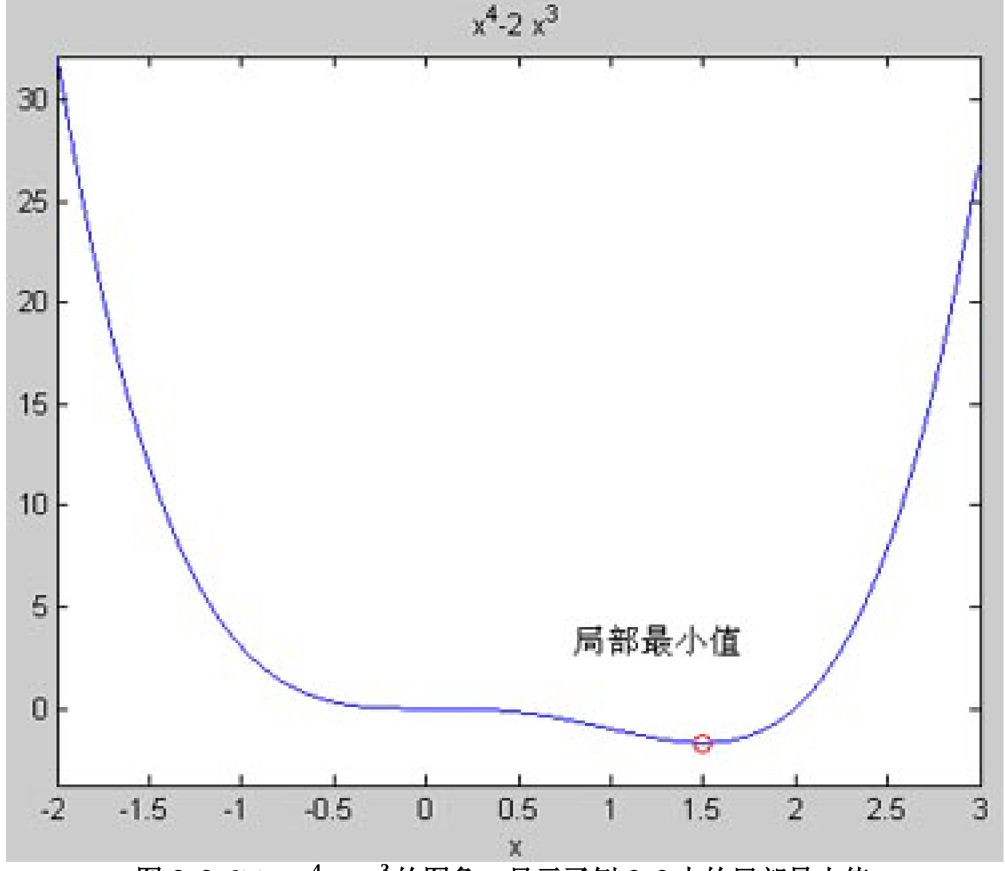


图 6-6 $f(x) = x^4 - 2x^3$ 的图象，显示了例 6-6 中的局部最小值

dsolve 命令

在 **MATLAB** 中我们可以使用 *dsolve* 命令求解符号微分方程。使用 *dsolve* 求解某个方程的语法是 *dsolve('equ')*，其中 *equ* 是用来表示方程的字串。这个命令会返回一个带有任意常量符号解，这些常量在 **MATLAB** 中表示为 *C*1、*C*2 等等。我们还能够为方程指定初始和边界条件，不管有多少条件，这些条件用逗号隔开并附带在“*equ*”后面，形式如 *dsolve('eqn', 'cond1', 'cond2', ...)*。

当使用 dsolve 时，导数用 D 指示，因此我们下面的方程：

$$\frac{df}{dt} = -f + \cos t$$

我们写成：

```
'Df = -2*f + cos(t)'
```

更高阶的导数我们通过在 D 后面带上阶数数字表示。因此要输入方程：

$$y'' + 2y' = 5\sin 7x$$

我们将写成：

```
'D2y + 2Dy = 5*sin(7*x)'
```

常微分方程(ODE)求解

我们通过学习一些简单的微分方程，以便对 **MATLAB** 处理 ODE 有个初步了解。对于一般的情况，我们可以使用 *dsolve* 命令求解，只需把方程传递给它即可。下面我们求解最基本的 ODE：

```
>> s = dsolve('Dy=a*y')
s =
C1*exp(a*t)
```

假设我们要绘制 *C*1 和 *a* 取不同值时的图象。我们可以给这些变量指定值，然后使用 *subs* 命令把它们赋给新标签：

```
>> C1 = 2; a = 4;
>> f = subs(s)
f =
2*exp(4*t)
```

初始条件用引号引起来放在方程后面。例如，假设：

$$\frac{df}{dt} = \frac{t}{t-5}\,y(t),\quad y(0) = 2$$

本例的 *dsolve* 命令格式是：

```
>> dsolve('Dy = y*t/(t-5)','y(0) = 2')
ans =
-2/3125*exp(t)*(t-5)^5
```

二阶或更高阶方程的处理情况相类似。例如：

$$\frac{d^2f}{dt^2} - y = 0,\quad y(0) = -1,\quad y'(0) = 2$$

以下面的形式输入 **MATLAB**：

```
>> dsolve('D2y - y = 0','y(0) = -1','Dy(0) = 2')
ans =
-3/2*exp(-t)+1/2*exp(t)
```

例 6-7

求下列方程的解并绘制 $0 \leq t \leq 10$ 范围内的图象

$$\frac{df}{dt} = t + 3, \quad y(0) = 7$$

解 6-7

这个方程通过调用一次 *dsolve* 即可轻易得解：

```
>> s = dsolve('Dy = t + 3','y(0) = 7')
s =
1/2*t^2+3*t+7
```

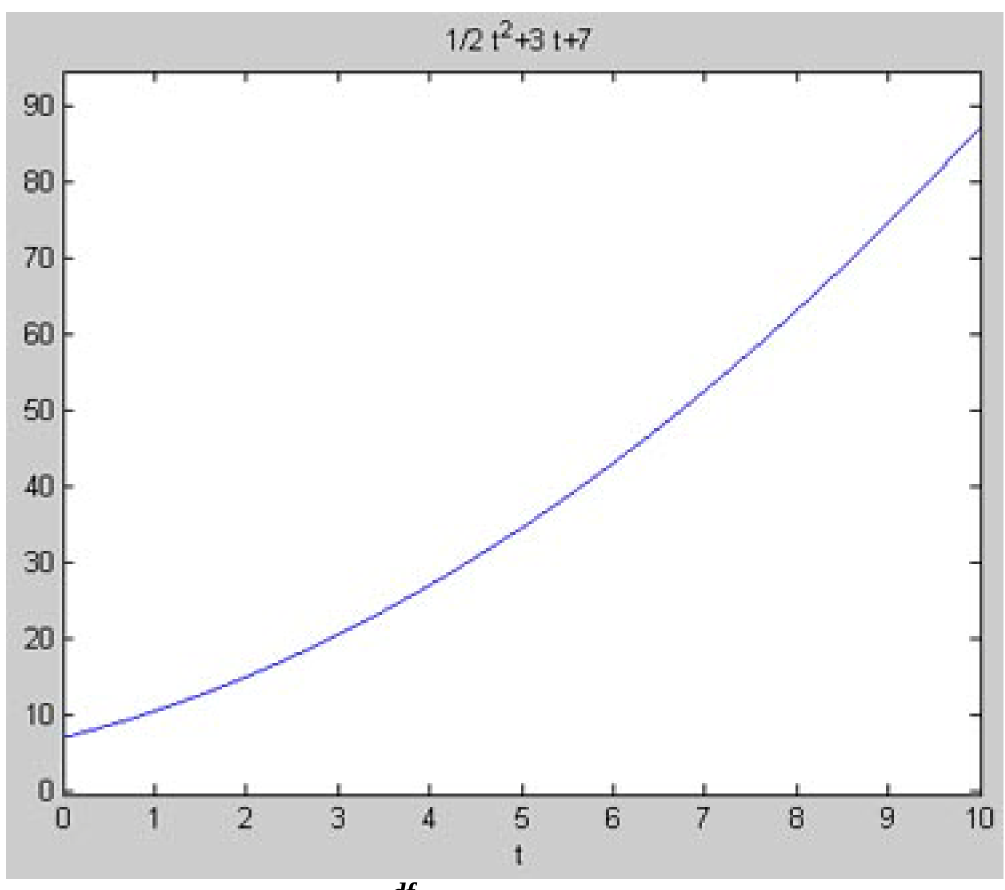


图 6-7 IVP $\frac{df}{dt} = t + 3, \quad y(0) = 7$ 解的图象

现在调用 *ezplot* 来产生图象：

```
>> ezplot(s,[0 10])
```

结果如图 6-7 所示。

例 6-8

求下面方程的解并绘制图象及它的渐近线。

$$\frac{dy}{dt} = y^2, \quad y(0) = 1$$

解 6-8

让 **MATLAB** 的 *dsolve* 为我们做这项困难的工作。我们求得解为：

常微分方程(ODE)求解



```
>> s = dsolve('Dy = y^2','y(0) = 1')
s =
-1/(t-1)
```

渐近线落在：

```
>> d = -1/s
d =
t-1

>> roots = solve(d)
roots =
1
```

现在我们绘制并保持：

```
>> ezplot(s)
>> hold on
```

现在绘制渐近线：

```
>> plot(double(roots)*[1 1], [-2 2],'--')
>> hold off
```

结果如图 6-8 所示。

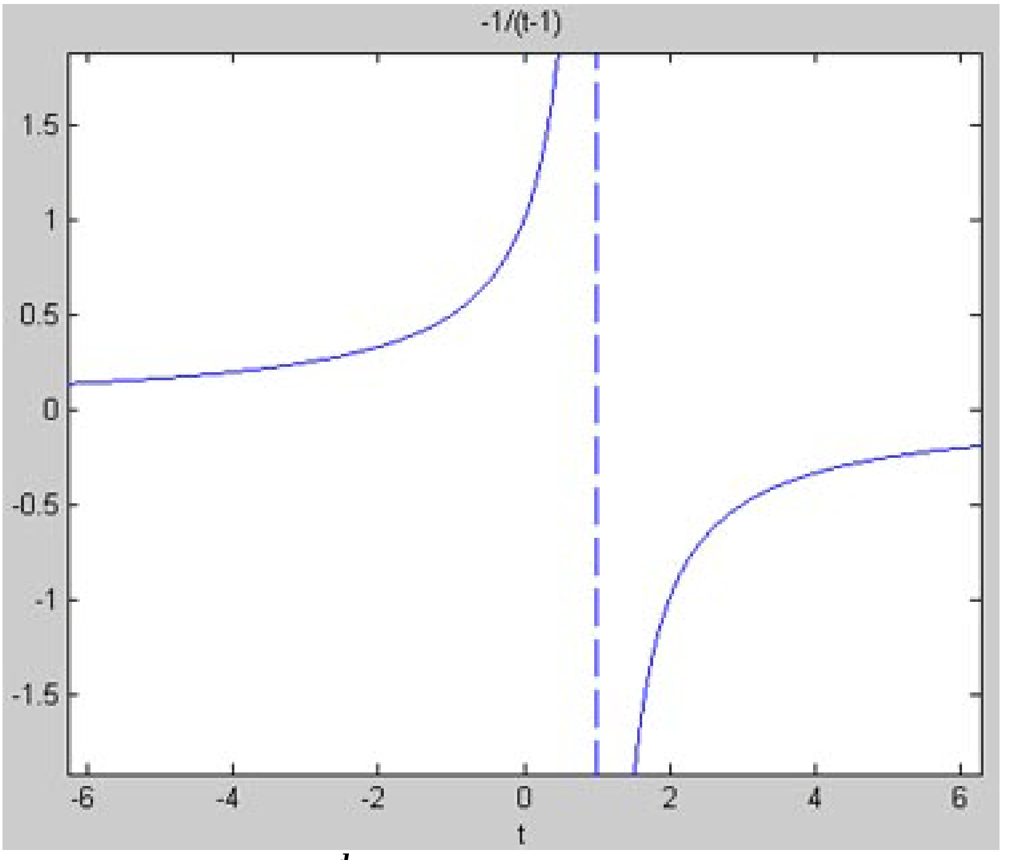


图 6-8 方程$\frac{dy}{dt}=y^2$，$y(0)=1$ 的解及其渐近线图象

例 6-9

求解下式并绘制-50≤x≤50 内的图象：

$$\frac{d^2f}{dx^2} - \frac{\sin x}{x}\left(1 - \frac{2}{x^2}\right) - \frac{2\cos x}{x^2} = 0,\quad f(0) = 2,\quad f'(0) = 0.$$

解 6-9

默认地，*dsolve* 使用 *t* 作为独立变量。我们可以告诉它使用其它变量——在命令行的末尾附带上我们要使用的独立变量。因此，调用 *dsolve* 的格式是：

```
>> g = dsolve('D2f - sin(x)/x-2*cos(x)/x^2+2*sin(x)/x^3 = 0','f(0)=2','Df(0)=0','x')
```

指定的独立变量放在边界条件后面声明。返回的解是：

```
g =
-sin(x)/x+3
```

我们绘制的函数图象，如图 6-9 所示。

例 6-10

求方程的通解并在同一个图形上绘制-1< *t* <1 上绘制 *C*1＝0,10,20,30 的图象：

$$\frac{dy}{dt} = -\frac{y}{\sqrt{1-t^2}}$$

解 6-10

使用 *dsolve* 很容易就得到解：

```
>> s = dsolve('Dy = -y/sqrt(1-t^2)')
s =
C1*exp(-asin(t))
```

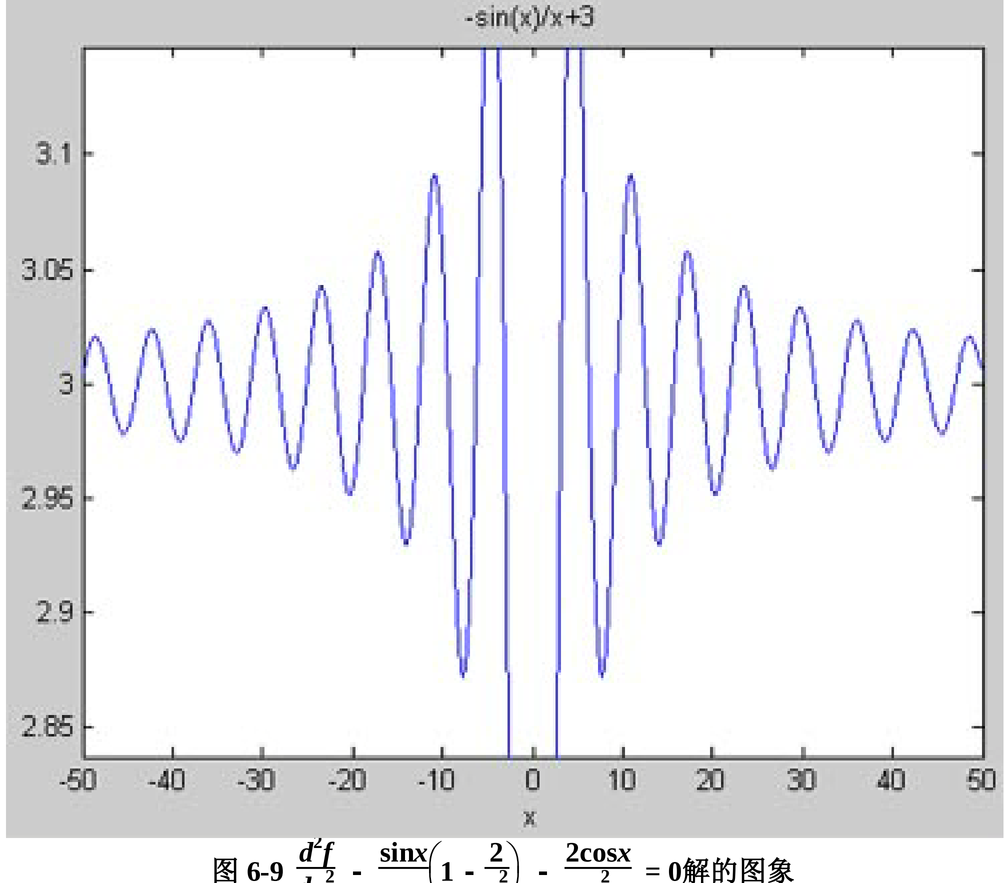


图 6-9 $\frac{d^2f}{dx^2} - \frac{\sin x}{x}\left(1 - \frac{2}{x^2}\right) - \frac{2\cos x}{x^2} = 0$解的图象

我们使用 *for* 循环在同一图形上绘制 *C*1 取不同值时的曲线。我们创建一个循环索引 *i* 让它取值 0,10,20 和 30，然后我们使用 *subs* 命令代替 *i* 的当前值并绘制结果图象，*for*



循环如下：

```
>> for i=0:10:30
f = subs(s,'C1',i);
ezplot(f,[-1,1])
hold on
end
```

第一行设置了 *i*=0,10,20,30 循环，语法是 *i=start*:*increment*:*finish*，接着我们使用 *subs* 命令告诉 **MATLAB** 把 C1 代换成当前循环值：

```
f = subs(s,'C1',i);
```

然后我们使用 *ezplot* 把曲线绘到图上，使用 *hold on* 告诉 **MATLAB** 我们要在每一次循环时在同一图形上绘制。因此每一次我们绘制的函数是

```
0, 10*exp(-asin(t)), 20*exp(-asin(t)), 30*exp(-asin(t))
```

end 语句是 *for* 循环的结束标记。接着我们调用 *hold off* 结束当前画图，然后给图象加上标题：

```
>> hold off
>> title('IVP 解')
```

结果如图 6-10 所示。

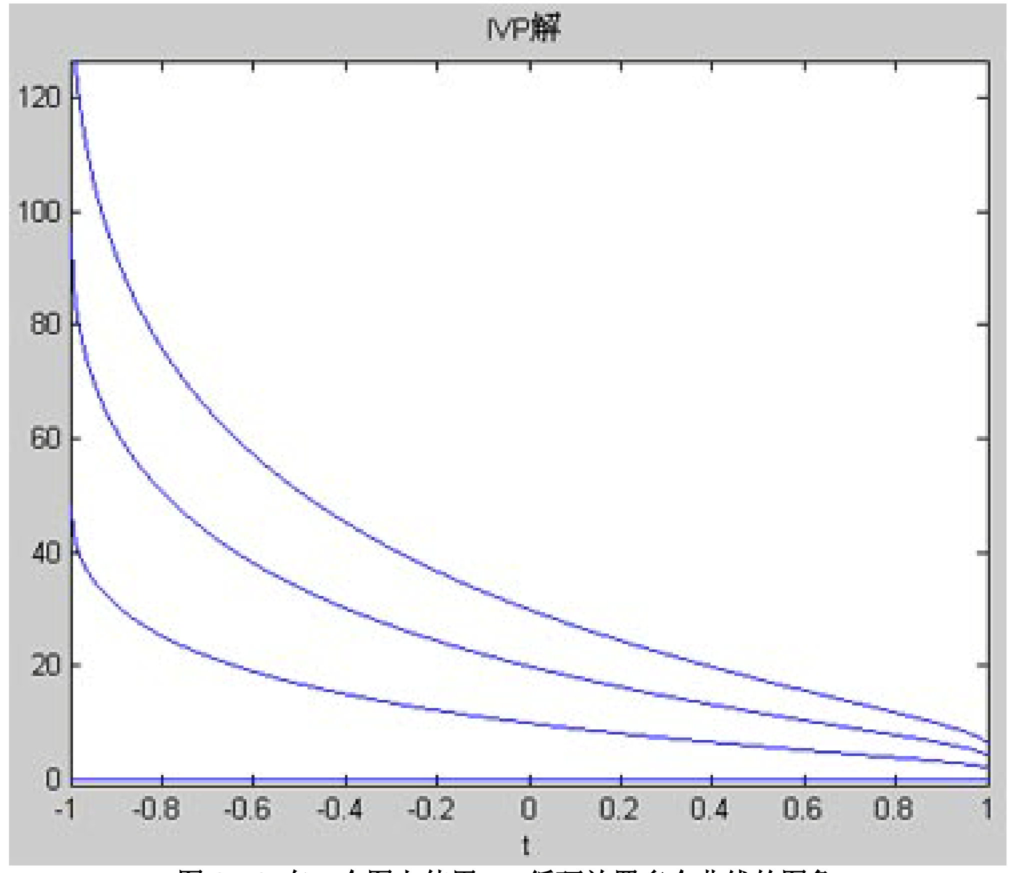


图 6-10 在一个图上使用 for 循环放置多个曲线的图象

例 6-11

求方程的解：

$$\frac{dy}{dt} = -2ty^2$$

以不同的初始值绘制解的图象，设 y(0) = 0.2, ..., 2.0，增量取 0.2。

解 6-11

首先我们求解方程：

```
>> f = dsolve('Dy=-2*t*y^2','y(0)=y0')
f =
1/(t^2+1/y0)
```

我们已经告诉 **MATLAB** 为 *y0* 设置初始值。现在我们可以写一个 *for* 循环把 y(0)的值设为 0.2,...,2.0。首先我们定义 *for* 循环及循环变量，指定其初始值、增量和终止值。

```
>> for i = 0.2:0.2:2
```

现在我们让 **MATLAB** 把解中的 *y0* 代换成 *i*：

```
>> temp = subs(f,'y0',i);
```

接着我们绘制图象：

```
>> ezplot(temp)
```

最后，我们告诉 **MATLAB** 保持现状（*hold on*）以便下一次循环可以在同一个图上绘制曲线，最后结果循环：

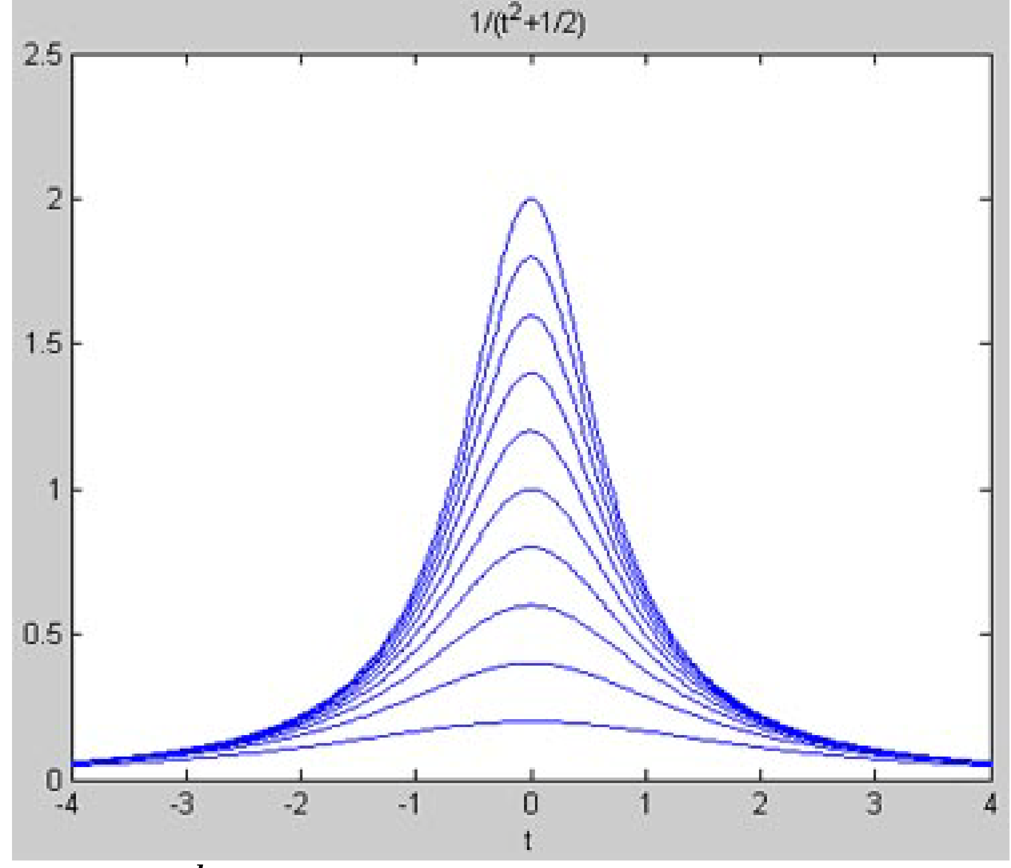


图 6-11 $\frac{dy}{dt} = -2ty^2$ 解的图象，初始条件是 y(0)=0.2,...,2.0，增量取 0.2

```
>> hold on
```



```
>> end
```

在循环结束后，我们可以使用 *axis* 命令把坐标值设置在我们希望的范围：

```
>> axis([-4 4 0 2.5])
>> hold off
```

不要忘了调用 *hold off*，以便 **MATLAB** 停止在同一个图上绘图。所有的这些运行的结果如图 6-11 所示。

方程组和相平面图

在这一节中我将学习质量弹簧系统方程及如何根据方程的解绘出相平面图。首先，我们如何使用 **MATLAB** 求微分方程组的解呢？答案是把每个方程传递给 dsolve。考虑下面的简单系统：

$$\frac{dX}{dt} = Y,\ \frac{dY}{dt} = -X$$
$$X(0) = -1,\ Y(0) = 2$$

输入方程组并求解的命令是：

```
>> s = dsolve('DX = Y','DY = -X','X(0)= -1','Y(0)=2');
```

方程组的解以向量的形式返回，我们可以提取它们：

```
>> s.X
ans =
-cos(t)+2*sin(t)
>> s.Y
ans =
sin(t)+2*cos(t)
```

我们可以把这两个解在图象上显示出来：

```
>> ezplot(s.X)
>> hold on
>> ezplot(s.Y)
>> hold off
```

结果如图 6-12 所示。这没有多大的帮助，因为我们无法确定哪条曲线是哪个解的。

我们添加一些附加命令，以便阐明图象中的曲线。假设我们要把解 $X(t)$绘制成红色实线条，而解 $Y(t)$绘制成蓝色虚线条，在 *ezplot* 中如何做呢？

第一步很简单，我们使用 *set* 告诉 **MATLAB** 使用红色画第一条曲线。我们调用 *findobj* 命令告诉 **MATLAB** 查找图象中的线条，然后使用 *set* 命令为线条设置颜色：

```
>> ezplot(s.X),set(findobj('Type','line'),'Color','r')
```

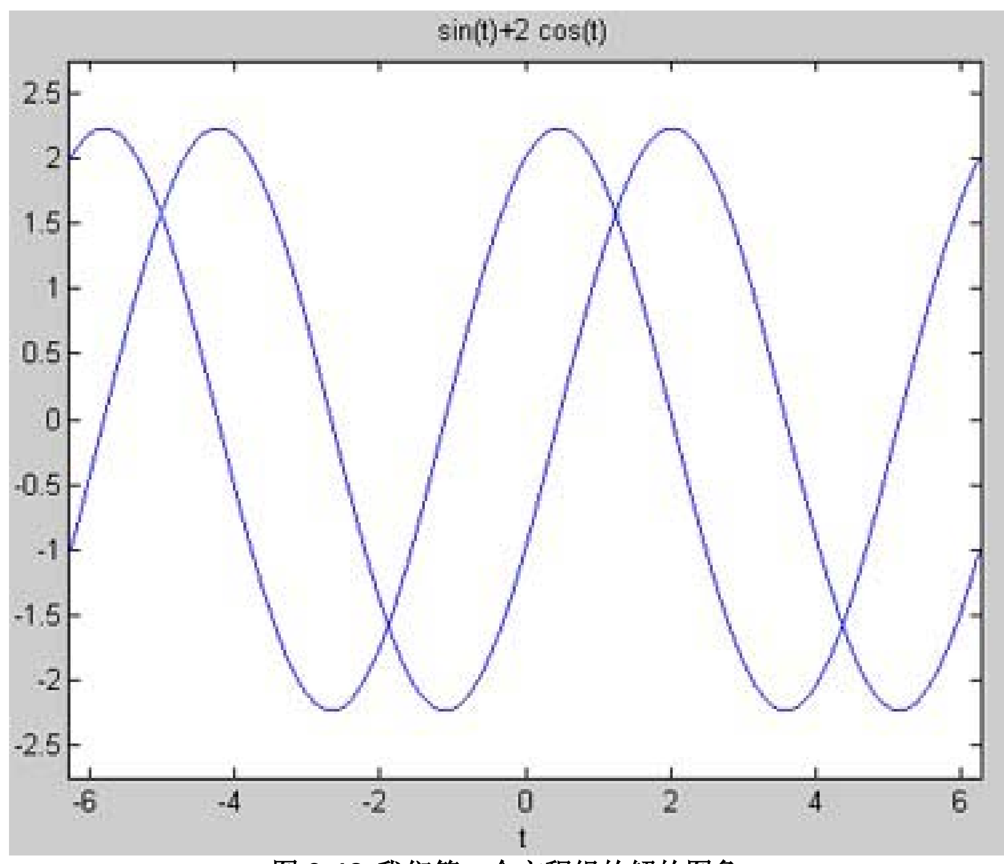


图 6-12 我们第一个方程组的解的图象

如果你试过这一行命令，你会看到一条红色的曲线很好地画在屏幕上。问题是这个命令告诉 **MATLAB** 把所有的线条变成红色——如果有多条线条存在的话。因此要把这条线条变成红色而把第二条绘成虚线，我们必须试些别的命令。首先，我们告诉 **MATLAB** 我们要在同一图上绘制另一条曲线：

```
>> hold on
```

现在我们添加第二条曲线：

```
>> ezplot(s.Y)
```

这将添加一条蓝色的曲线。我们要得到第二条曲线的引用以便我们可以让 **MATLAB** 改变它。接着我们调用 `get` 命令，它将返回当前图形对象的句柄：

```
>> h=get(gca,'children');
```

现在我们可以使用下面的命令改变它了：

```
>> set(h(1),'linestyle','--')
```

结果如图 6-13 显示。



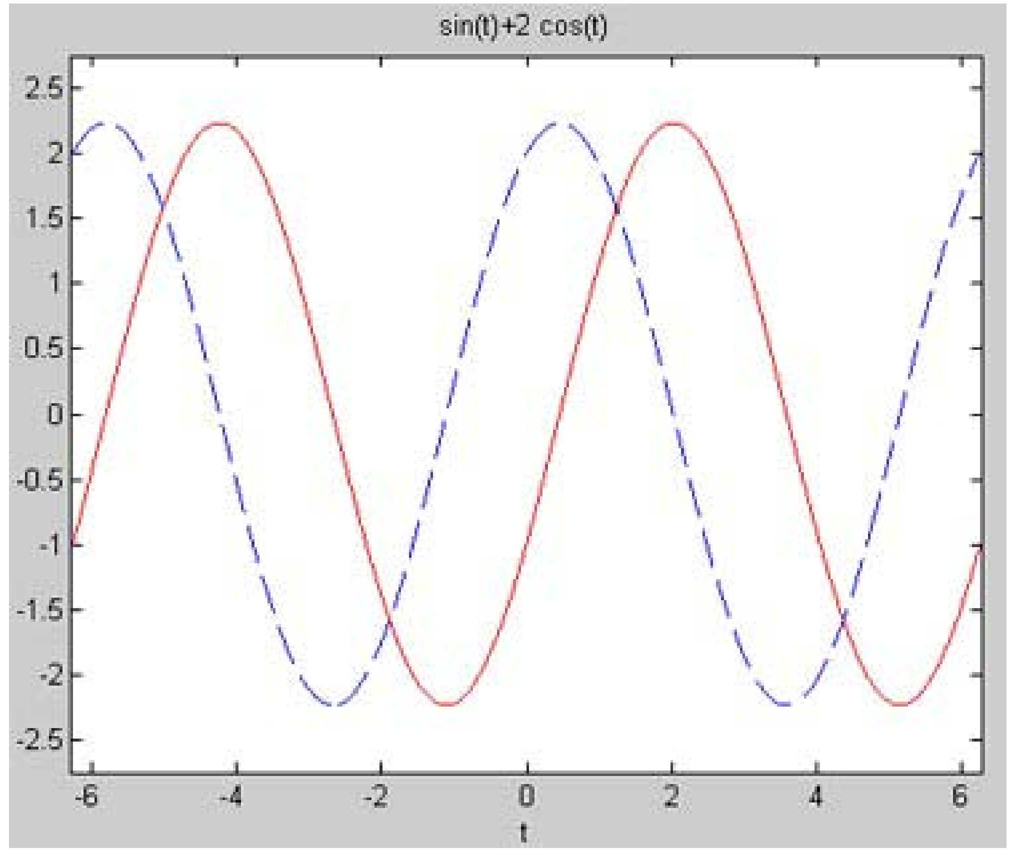


图 6-13 使用 *ezplot* 并为不同的曲线设置不同的风格

现在我们考虑一个质量弹簧系统并且看看如何得到它的位置解、动量并绘制相图。

例 6-12

用 $x(t)$ 表示质量弹簧系统的质量所在位置，$p(t)$ 表示它的动量。并假设这种情况遵循阻尼振荡方程：

$$2x'' + x' + 8x = 0,$$

$$\frac{dp}{dt} = -p - 8x$$

$$x(0) = 2,\ p(0) = x'(0) = 0$$

求解 $x(t)$ 和 $p(t)$，并绘制系统的相图。

解 6-12

首先我们调用 *dsolve* 求得方程的解：

```
>> s = dsolve('2*D2x+Dx+8*x = 0','Dp = -p - 17*x','x(0)=4',' Dx(0)=0','p(0)=0')
```

MATLAB 以下面的形式给出结果：

```
s =
    p: [1x1 sym]
    x: [1x1 sym]
```

我们输入 *s.x* 和 *s.p* 即可访问方程的解。让我们看看这个系统中位置与时间的关系函数：

```
>> s.x
ans =
exp(-1/4*t)*(4/21*sin(3/4*7^(1/2)*t)*7^(1/2)+4*cos(3/4*7^(1/2)*t))
```

对于动量，我们得到：

```
>> s.p
ans =
-68/9*cos(3/4*7^(1/2)*t)*exp(-1/4*t)-748/63*7^(1/2)*exp(-1/4*t)*sin(3/4*7^(1/2)*t)+68/9*exp(-t)
```

现在我们绘制位置随时间变化的函数图象。我们使用：

```
>> ezplot(s.x,[0 10])
>> title('质量的位置')
```

正如从图 6-14 所看到的，质量的位置按指数衰减的规律描述出来，这正是阻尼系统所期望的。

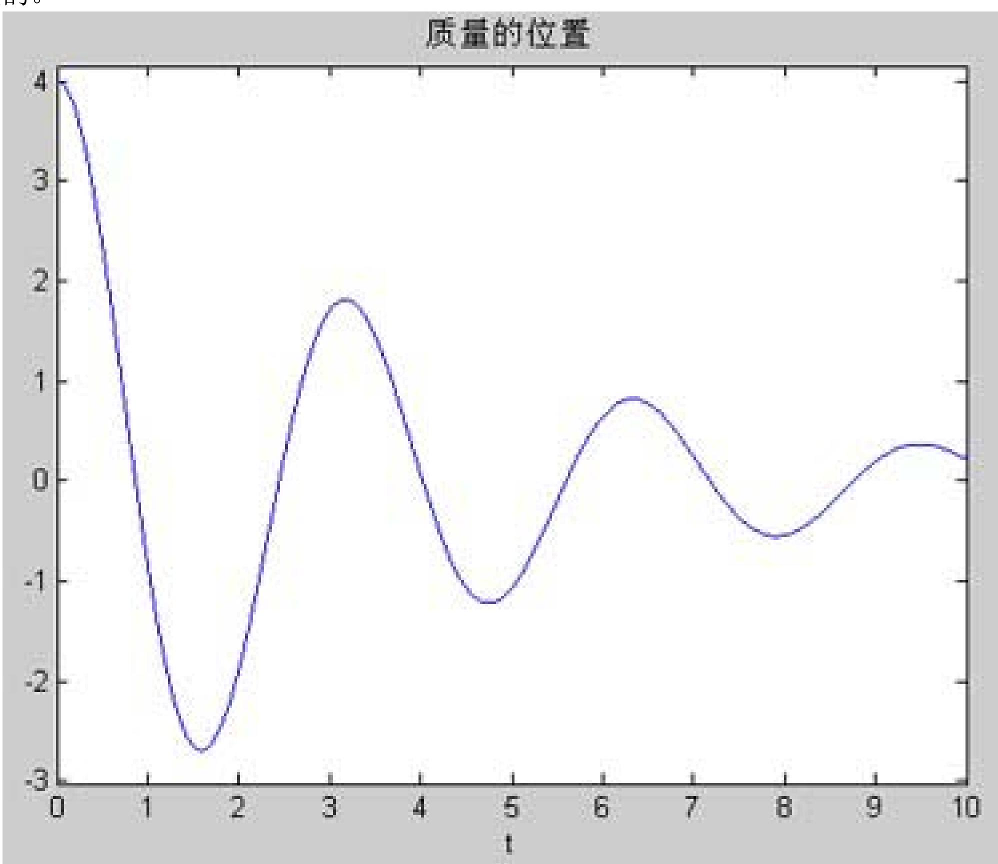


图 6-14 方程 $2x'' + x' + 8x = 0$ 所描述的衰减振荡

现在我们绘制相同时间内的动量：

```
>> ezplot(s.p,[0 10])
>> title('动量')
```

如图 6-15 所示，动量也是一个衰减振荡，不过振幅大了很多。

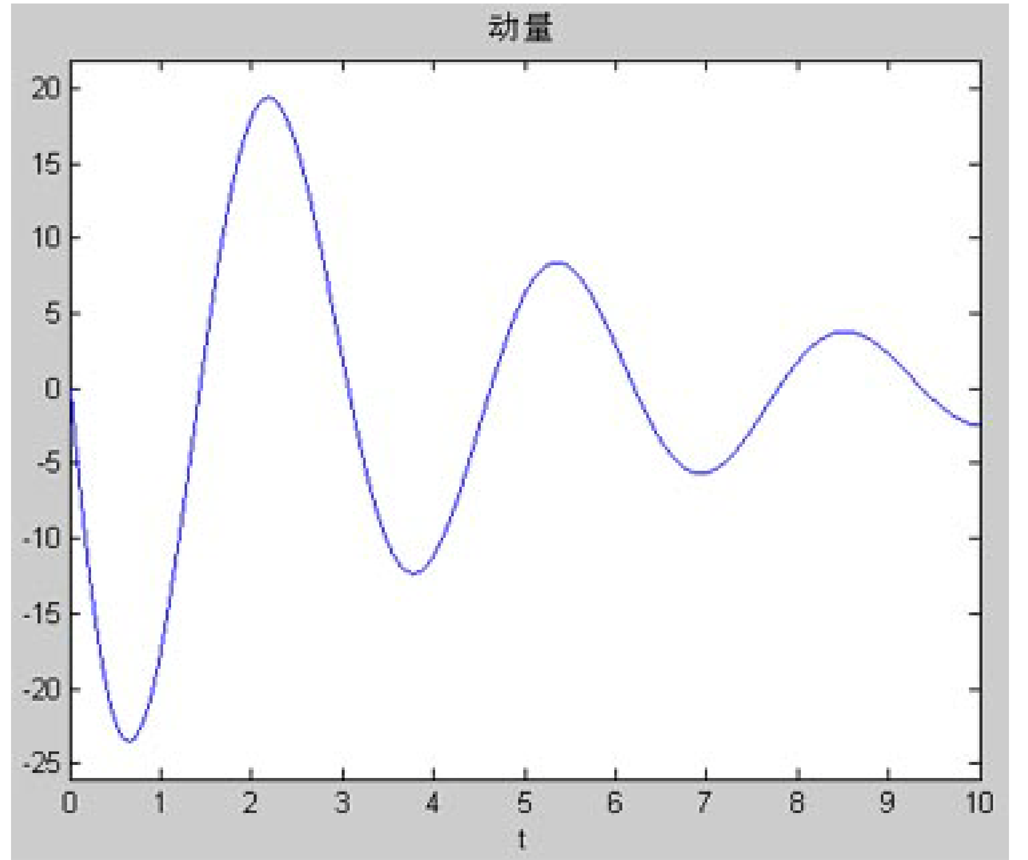


图 6-15 方程 $2x'' + x' + 8x = 0$ 所描述的系统动量

相图是一个 *p-x* 图象，我们可以尝试用下面的方式调用 *ezplot*。我们可以同时传递两个函数给它：

```
>> ezplot(s.x,s.p,[-5 5])
```

我们修改一下坐标轴的范围，然后添加一个标题：

```
>> axis([-8 8 -25 20])
>> title('参量图')
```

结果如图 6-16 所示。

你可能觉得这个图看起来有点奇怪，让我们来试一下其它内容，也可以产生一个数值参量图。首先我们定义一个时间间隔数值：

```
>> tvalues = (0:0.1:10);
```

现在我们使用 *subs* 在这个时间间隔内产生表示位置与动量函数的数值：

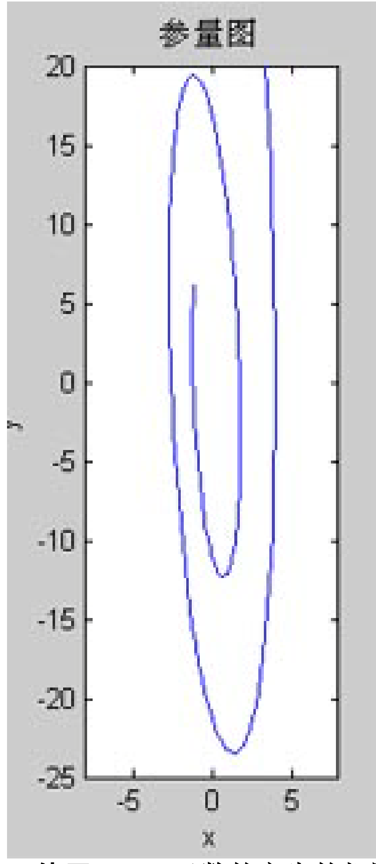


图 6-16 使用 *ezplot* 函数的产生的相图

```
>> xval = subs(s.x,'t',tvalues);
>> pval = subs(s.p,'t',tvalues);
```

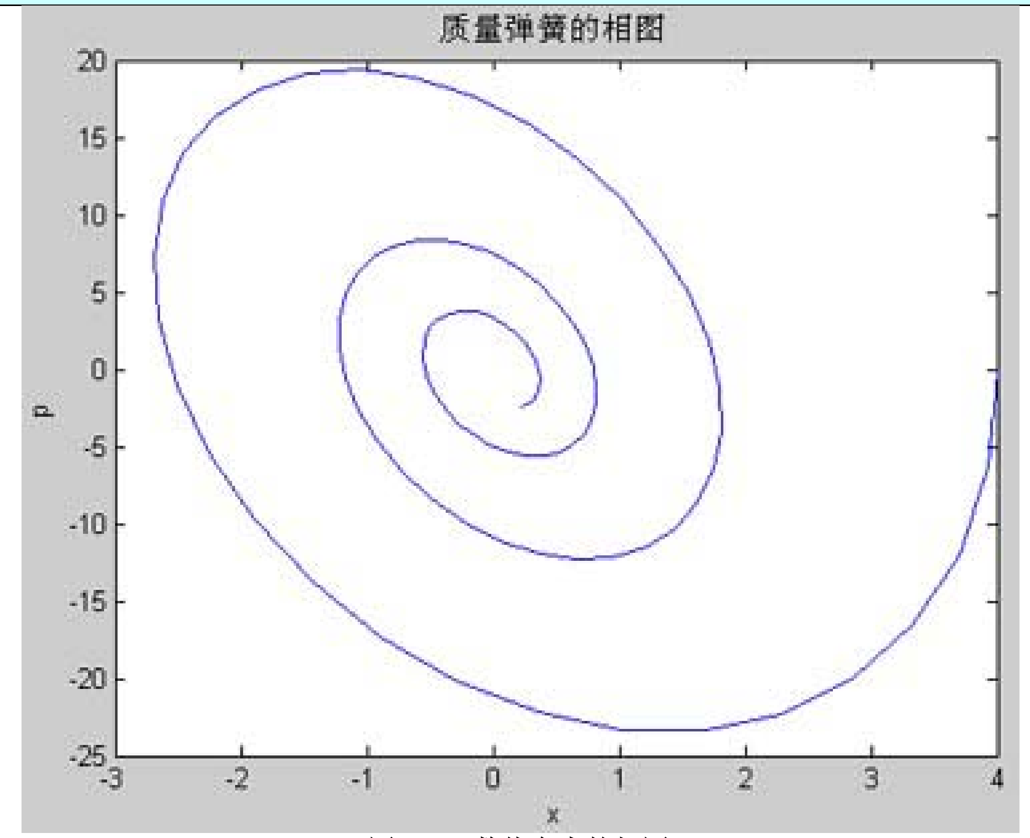


图 6-17 数值产生的相图

现在我们用下面的形式调用 plot 产生相图：

```
>> plot(xval,pval),xlabel('x'),ylabel('p'),title('质量弹簧的相图')
```

结果好多了，如图 6-17 所示。

例 6-13

求临界阻尼系统的解：

$$\frac{d^2x}{dt^2}+\frac{dx}{dt}+\frac{1}{4}x=0,\quad \frac{dp}{dt}=-\frac{1}{2}p-\frac{1}{4}x,\quad x(0)=4,\quad p(0)=0$$

解 6-13

我们输入方程并求解：

```
>> s = dsolve('D2x + Dx + (1/4)*x = 0','Dp + (1/2)*p + (1/4)*x = 0','x(0) = 4','Dx(0)=0','p(0)=0');
```

结果是：

```
>> s.x
ans =
exp(-1/2*t)*(4+2*t)

>> s.p
ans =
-1/8*(8*t+2*t^2)*exp(-1/2*t)
```

我们绘制位置随时间变化的函数图象，如图 6-18：

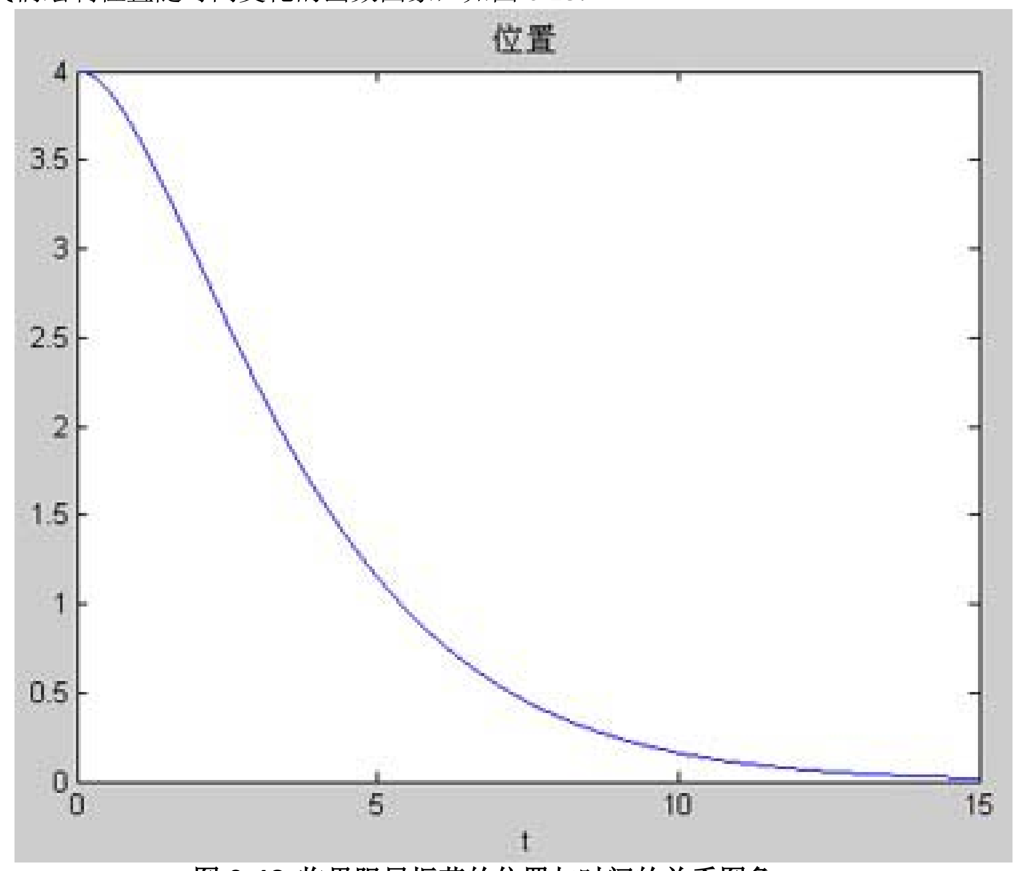


图 6-18 临界阻尼振荡的位置与时间的关系图象

```
>> ezplot(s.x,[0 15])
>> axis([0 15 0 4])
>> title('位置')
```

现在我们产生动量的图象：

```
>> ezplot(s.p,[0 15])
>> title('动量')
```

动量的图象如图 6-19 所示。注意动量和质量的位置没有经过振荡很快地被抑止，这正是临界阻尼的情况。

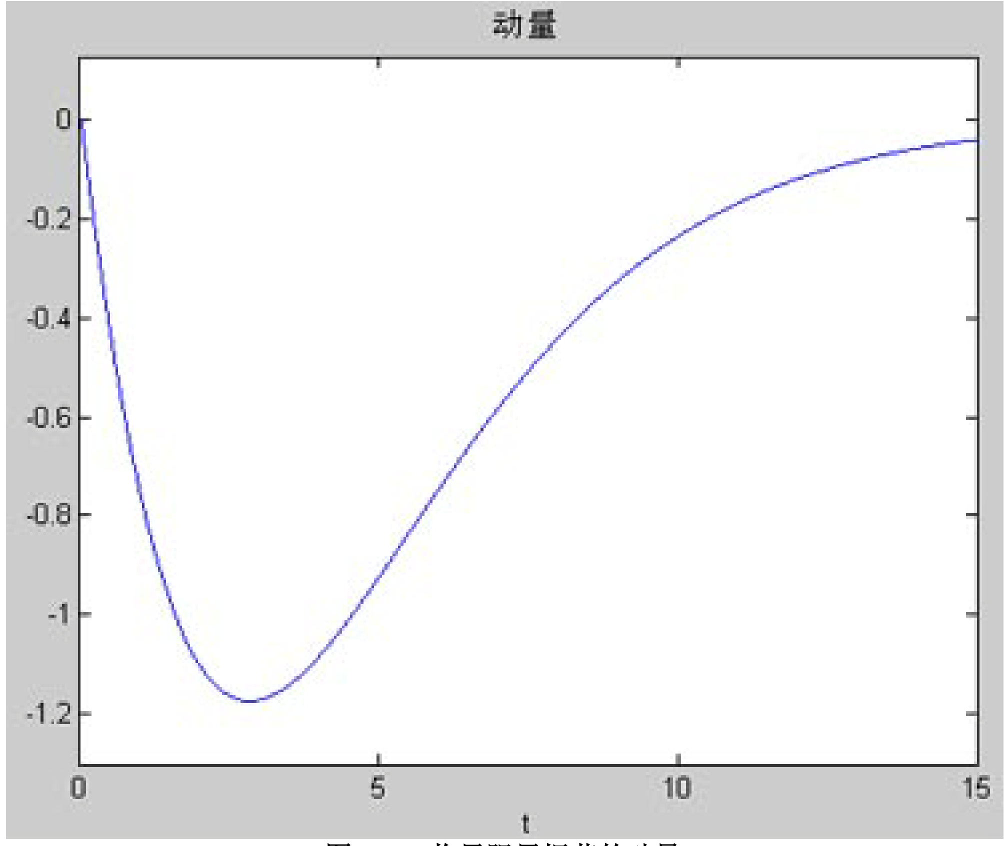


图 6-19 临界阻尼振荡的动量

习题

1. 计算 $\lim_{x\to 0}\frac{\sin 4x}{x}$ 。
2. $\lim_{x\to 2}\frac{x-2}{|x-2|}$ 的极限存在吗？
3. 找出 $\frac{x}{x^2-3x}$ 的渐近线，并把它用虚线在函数的图象上表示出来。
4. 计算 $\lim_{x\to\pi/2}\frac{\cos\theta}{1+\sin\theta}$。
5. 设函数 $f(x)=\frac{1}{3x^2+1}$

   a)函数的一阶和二阶导数是什么？



b)函数的临界点是什么？

c)f''在这些临界点上的值是多少？

d)它们是局部的最小值和最大值吗？在图上绘制该函数的图象。

6. 使用 **MATLAB** 指出 $y = 2\sin t - 3\cos t$ 不满足方程

$$y'' - 11y = -4\cos 6t$$

7. 求方程$\frac{dx}{dt} = -2x + 8$ 的解并在同一个图上绘制常量 $C1=1,2,...,5$ 时的图象。

8. 求 $y'' - y' - 2y = 2t,\ y(0) = 0,\ y'(0) = 0$ 的解。

9. 解方程组 $x'' + x = 0,\ p' + p + x = 0,\ x(0) = 4,\ x'(0) = p(0) = 0$。

10. 解方程组 $x'' + 2x' + x = 0,\ p' = x,\ x(0) = 1,\ x'(0) = p(0) = 0$ 并绘制相图。

【参考答案在第 231 页】



第七章

常微分方程（ODE）的数值解

在上一章中我们学习了如何求解符号 *ODE*（常微分方程）。在这一章我们将学习如何使用 **MATLAB** 求常微分方程的数值解。在后面的内容我们会看到，**MATLAB** 有几个用来解微分方程的解算器。



使用 *ODE23* 和 *ODE45* 求解一阶方程

要求解 *ODE* 的数值解，首先得定义表示方程的的函数。我们的第一个例子是考虑下面的方程：

$$\frac{dy}{dt} = \cos(t)$$

我们很容易求得方程的积分为 $y(t) = \sin(t) + C$，它将可以用来检验我们得到的数值解。首先我们定义函数，我们创建一个.*m* 文件，输入下面的内容。

```
function ydot = eq1(t,y)
ydot = cos(t);
```

我们通过调用 *ODE32* 函数来求解 *ODE*。这个函数使用使用二阶和三阶 *Runge-Kutta*（龙格-库塔）法对微分方程进行积分。使用的语法是：

```
[t,y] = ode23('func_name', [start_time, end_time], y(0))
```

我们的函数称为 *eq1*。让我们求 $0 \le t \le 2\pi$ 内的解并假设 $y(0) = 2$。调用的语句是：

```
>> [t,y] = ode23('eq1',[0 2*pi],2);
```

由于这个方程比较简单，**MATLAB** 很快就解出答案。既然我们是在做数值计算，看结果我们就要把它绘制出来。首先我们产生一个表示解析解的数集，然后进行比较：

```
>> f = 2 + sin(t);
```

现在使用下面的命令绘制图象：

```
>> plot(t,y,'o',t,f),xlabel('t'),ylabel('y(t)'),axis([0 2*pi 0 4])
```

图象如图 7-1 所示。图中用圆圈标记的解是由 *ode32* 返回的，而实线则解析解，它与 *ode32* 返回的解相当接近。

有多接近呢？我们可以计算它们的相对误差，如果 $f(t)$表示解析解而 $y(t)$表示数值解，其相对误差由下式给出：

$$\left|\frac{f(t) - y(t)}{f(t)}\right|$$

我们可以把误差函数起名为 *err*。首先我们为储存误差点的数组分配内存空间，使用一个所有元素都为零的数组：

```
>> err = zeros(size(y));
```

现在我们使用 *for* 循环遍历数据，计算每个点上的相对误差：

```
>> for i = 1:1:size(y)
err(i) = abs((f(i)-y(i))/f(i));
end
```

结果是：

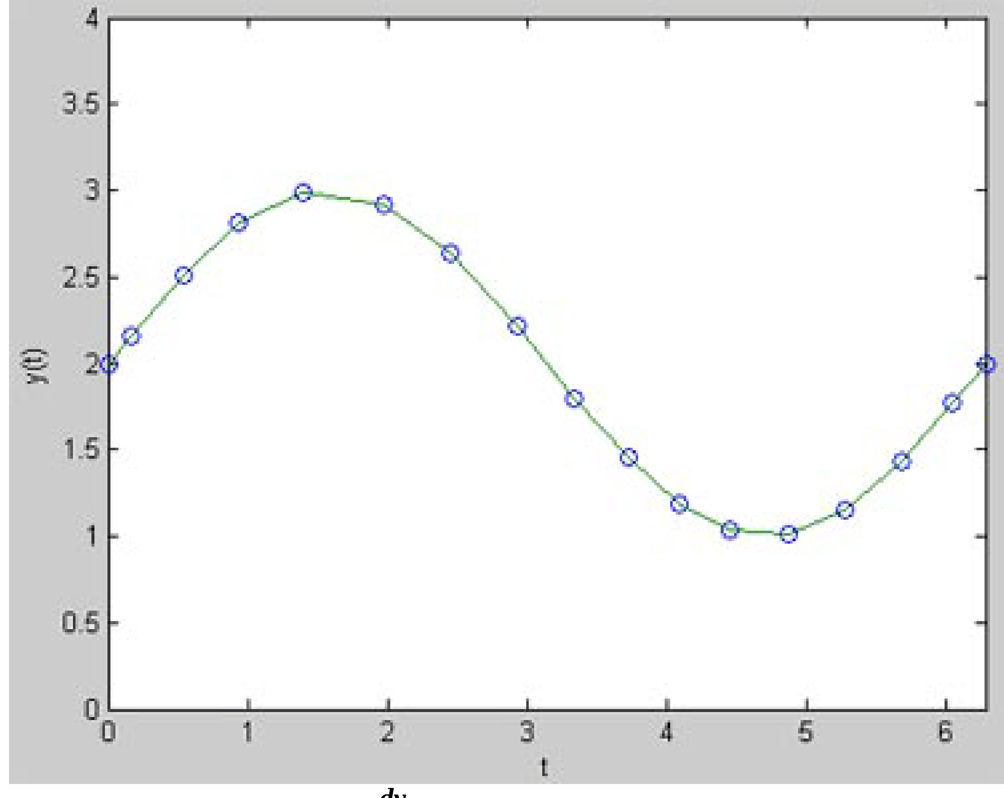


图 7-1 显示了$\frac{dy}{dt} = \cos(t)$ 解析解和数值解的图象

```
>> err
err =
  1.0e-003 *
         0
    0.0001
    0.0091
    0.0301
    0.0743
    0.2115
    0.2876
    0.3780
    0.4659
    0.5597
    0.6480
    0.6907
    0.6050
    0.4533
    0.3164
    0.2414
    0.2129
```

在这个例子中，误差非常小，其中最大的误差是：

```
>> emax = max(err)
emax =
  6.9075e-004
```

我们看一下由 *ode23* 解算器返回的数据点：

```
>> y
y =
    2.0000
    2.1593
    2.5110
    2.8080
    2.9848
    2.9169
    2.6344
    2.2123
    1.7999
    1.4512
    1.1892
    1.0323
    1.0115
    1.1536
    1.4337
    1.7689
    1.9996
```

相对误差已经足够小了，我们对这个结果的精度可以满意了。

ode45 函数使用更高阶的 *Runge-Kutta* 公式。让我们看看它与我们刚才使用的 *ode32* 有什么不同。调用的格式相类似：

```
>> [t,w] = ode45('eq1',[0 2*pi],2);
```

然而，解法计算了更多数量的点。因此我们必须重新产生解析解，然后在一个图中一同绘制它们：

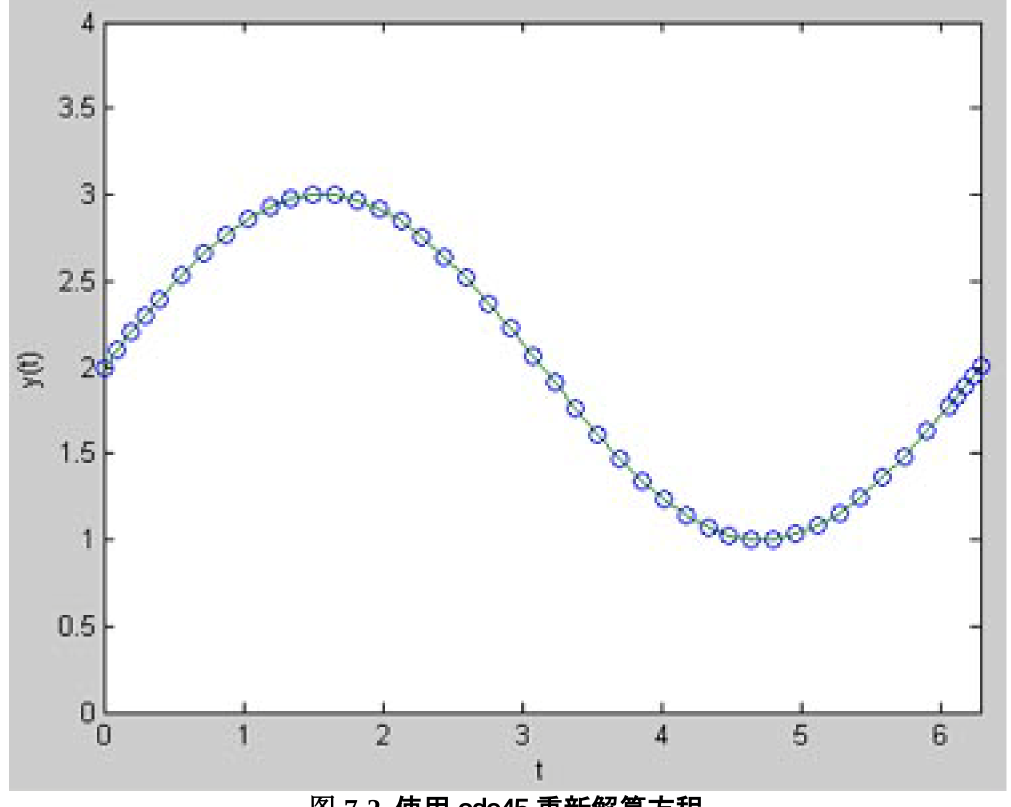


图 7-2 使用 ode45 重新解算方程

```
>> f = 2 + sin(t);
```

```
>> plot(t,w,'o',t,f),xlabel('t'),ylabel('y(t)'),axis([0 2*pi 0 4])
```

结果如图 7-2 所示。你可以看到根据数值画出的点比前面我们从 *ode32* 得到的密度要高。

我们比较解的大小：

```
>> size(w),size(y)
ans =
    45     1
ans =
    17     1
```

ode45 解算器返回了 45 个数据点而 *ode23* 解算器只返回 17 个数据点。重要与否就取决于你的应用了。让我们创建另一个零数组并计算相对误差：

```
>> err = zeros(size(w));
>> for i = 1:1:size(w)
err(i) = abs((f(i)-w(i))/f(i));
end
```

这次我们再找出相对误差的最大值：

```
>> wmax = max(err)
wmax =
  4.9182e-006
```

不知你是否还记得，我们使用 *ode23* 解算器得到的最大误差是：

```
>> emax
emax =
  6.9075e-004
```

这要比 *ode45* 得到的相对误差大得多，实际上它差不多大 141 倍：

```
>> emax/wmax
ans =
  140.4473
```

如果误差对你非常重要，那么使用 *ode45* 将是明智的选择。下面我们看一个例子。

例 7-1

考虑下面带有扰动的常微分方程：

$$\frac{dy}{dt} = 5(F(t) - y)$$

其中的扰动方程是：

$$F(t) = te^{-t/tc}\cos(\omega t)$$

根据表 7-1 中所给的时间常量 tc 和频率 ω 求数值解并绘制它。

表 7-1 例 7-1 中所需要的时间常量和频率

时间常量	频率
0.01 s	628 rad/s
0.1 s	6.28 rad/s

解 7-1

为了便于使用几组不同的时间常量和频率，我们把这些量声明为全局变量。首先我们写一个函数实现这个方程：



```
function ydot = eq2(t,y)
global tc w
F = t*exp(-t/tc)*cos(w*t);
ydot = 5*(F-y);
```

遵循文本中的步骤，我们把这些代码写在一个*.m* 文件中并保存，接着我们在 **MATLAB** 定义全局变量（时间常量和频率）：

```
>> global tc w
```

现在我们把表中的第一组值赋给变量：

```
>> tc = 0.01; w = 628;
```

现在我们设置与初始条件一起计算的时间终点：

```
>> final_time = 0.1; y0 = 0;
```

接着我们调用 *ode45* 产生第一个解：

```
>> [t,y] = ode45('eq2',[0 final_time],y0);
```

我们把解绘制在图 7-3 中。

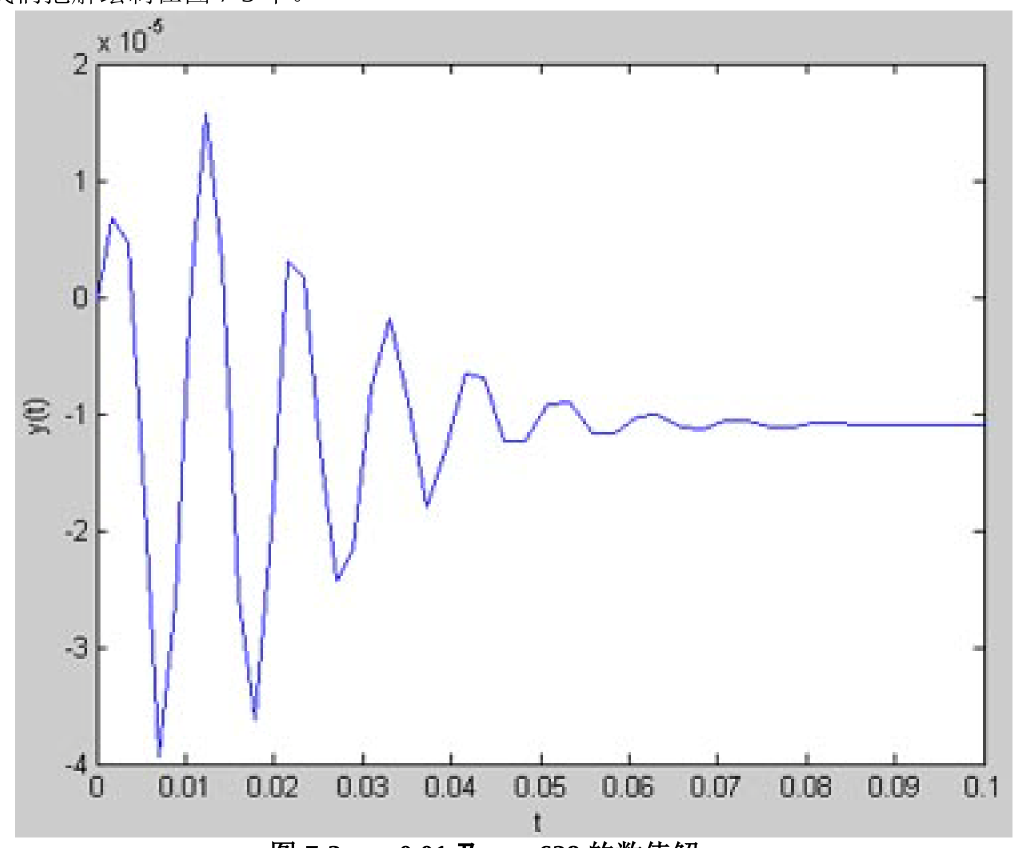


图 7-3 *tc* = 0.01 及 *w* = 628 的数值解

我们把它与扰动函数做比较：

```
>> plot(t,y,t,F,'--'),xlabel('t')
```

扰动函数用虚线表示出来，可以从图 7-4 看到。它比它的响应要大得多——但仍然有相似的函数形式。在这两个例子中我们都看到遵循指数衰减的振荡。

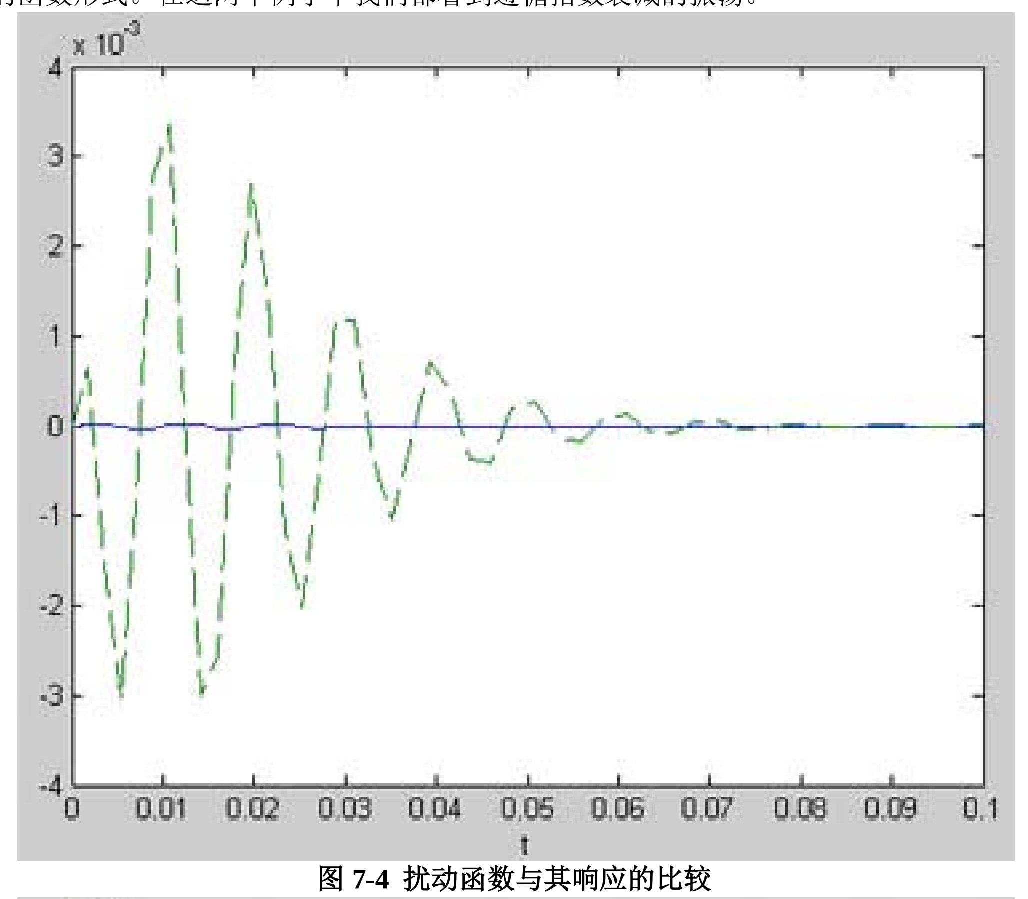


图 7-4 扰动函数与其响应的比较

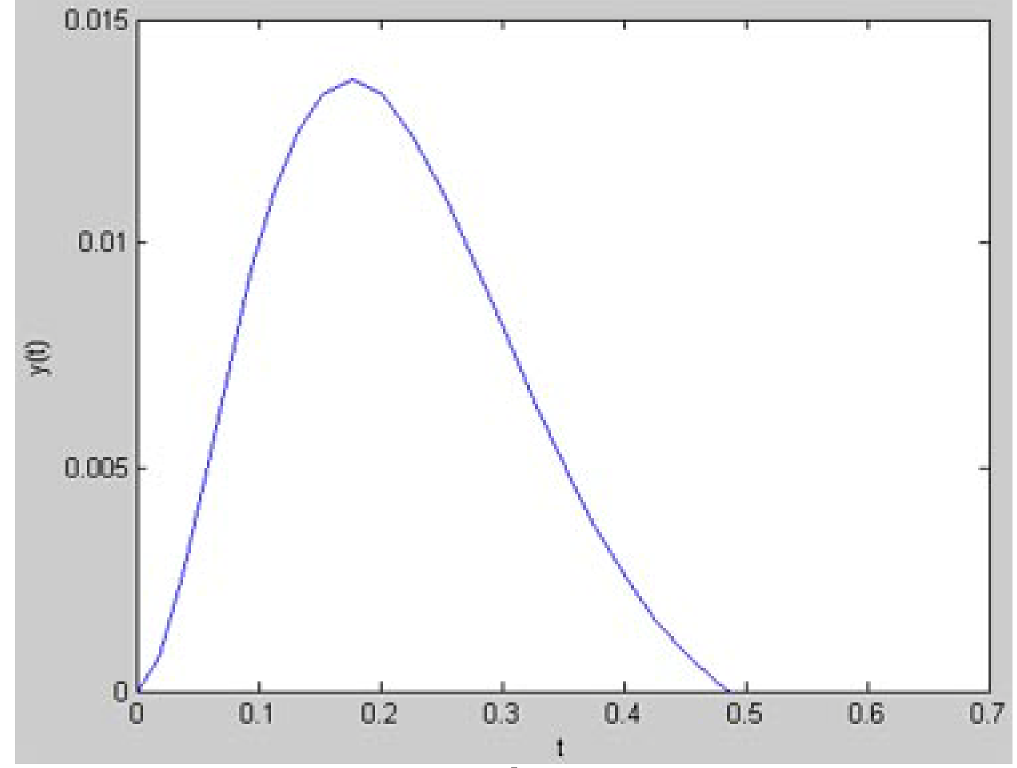


图 7-5 ω=6.28 和 tc=0.1 时 $\frac{dy}{dt} = 5(F(t) - y)$ 的响应图象

在这个例子中，扰动看来要比响应持续更久。

现在我们试用下一组数值。

```
>> tc = 0.1; w = 6.28;
```

由于时间常量大得多，我们把时间间隔设大一点：

```
>> final_time = 1.0;
```

现在我们调用解算器：

```
>> [t,y] = ode45('eq2',[0 final_time],y0);
>> plot(t,y),xlabel('t'),ylabel('y(t)'),axis([0 0.7 0 0.015])
```

本例中的图象如图 7-5 所示。

让我们把扰动函数及响应的图象画到同一个图上。在这种情况下，你可以很清楚地看到响应落后于扰动，如图 7-6 所示。以虚线条显示扰动函数的绘图命令如下：

```
>> plot(t,y,t,F,'--'),xlabel('t'),ylabel('y(t)'),axis([0 0.7 0 0.05])
```

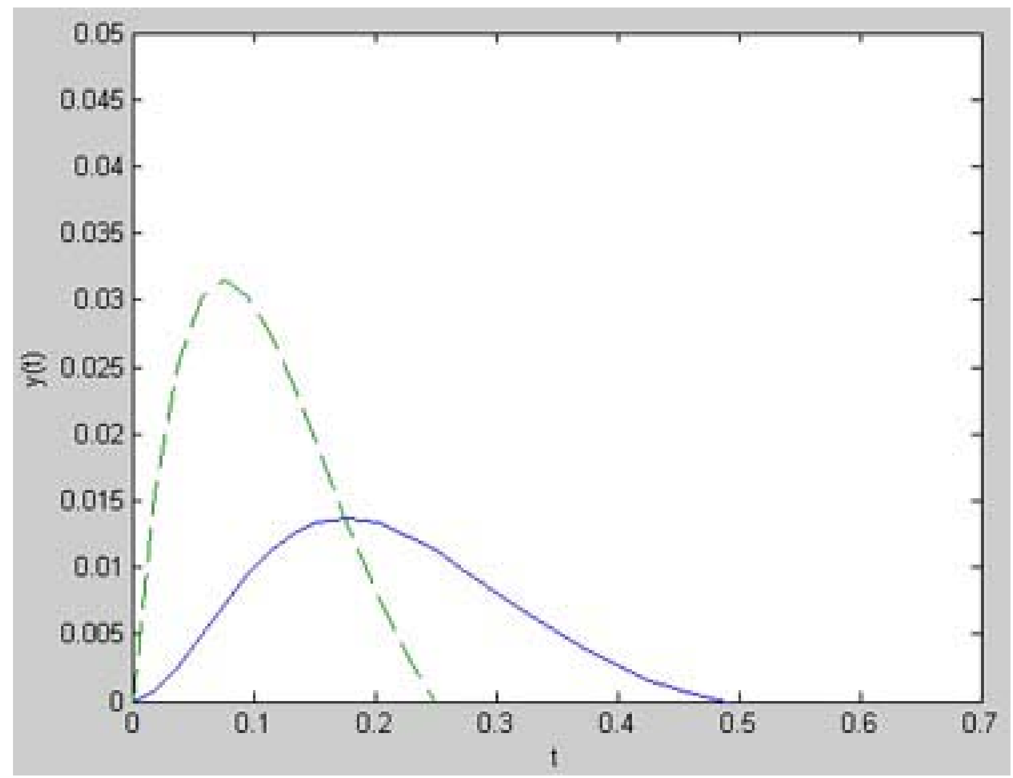


图 7-6 例 7-1 中检验的第二种情况，我们得到响应落后于扰动的图象，它们具有相类似的图象

二阶方程求解

现在我们看看如何求解二阶方程的数值解。这里使用的窍门是把一个方程分解为有两个方程的方程组。因此先让我们看看如何求解方程组：

例 7-2

求下列的方程组

$$\frac{dx}{dt} = -x^2 + y,\ \frac{dy}{dt} = -x - xy$$

初始条件是 $x(0) = 0$, $y(0) = 1$。绘制这两个函数的图象及相图。

解 7-2

首先我们像以前一样为右边的微分方程创建一个函数。当然，这一次是方程组，现在我们需要一个 2×1 的列向量。函数如下：

```
function xdot = eqx(t,x);
%创建数组储存数据
xdot = zeros(2,1);
xdot(1) = -x(1)^2 + x(2);
xdot(2) = -x(1) - x(1)* x(2);
```

一切顺利，相当简单。现在我们使用给定的初始条件用 *ode45* 求解：

```
>> [t,x] = ode45('eqx',[0 10],[0,1]);
```

现在我们绘制图象。使用 x(:,1)访问第一个函数，而第二个函数则写成 x(:,2)。绘图的命令是：

```
>> plot(t,x(:,1),t,x(:,2),'--'),xlabel('t'), axis([0 10 -1.12 1.12])
```

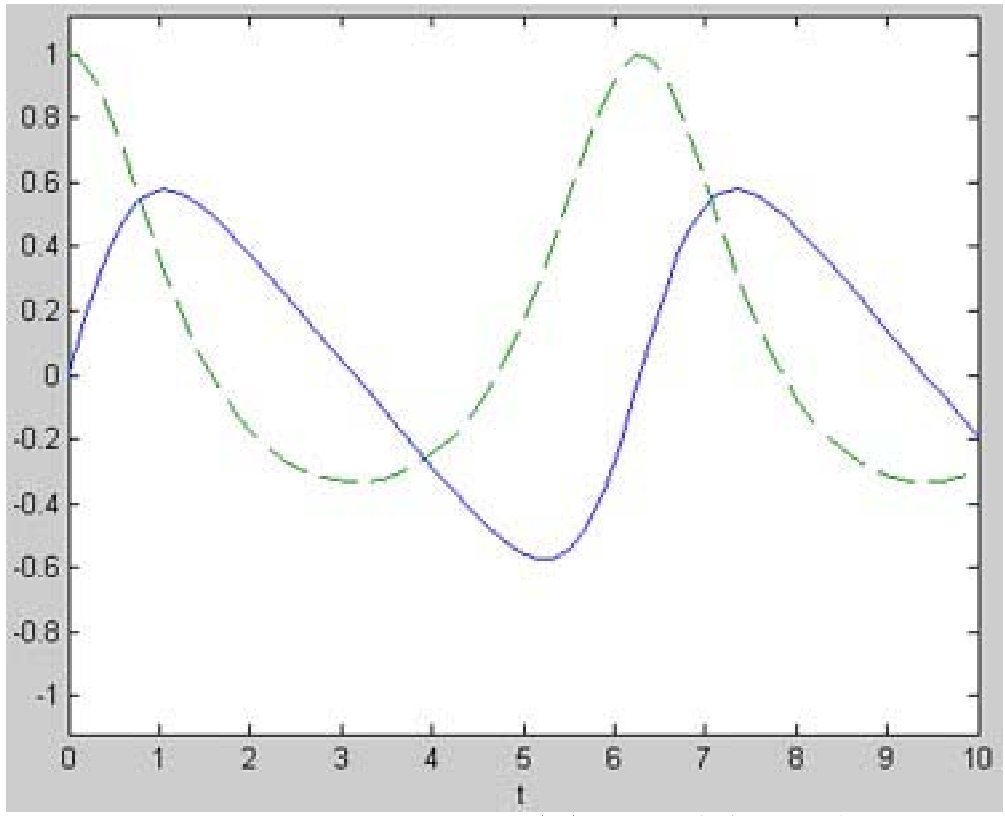


图 7-7 例 7-2 中方程组的解 *x*（实线）和 *y*（虚线）的图象

这两个函数的图象显示在图 7-7 中。绘制相图的命令是：

```
>> plot(x(:,1),x(:,2))
```

相图显示在图 7-8 中。

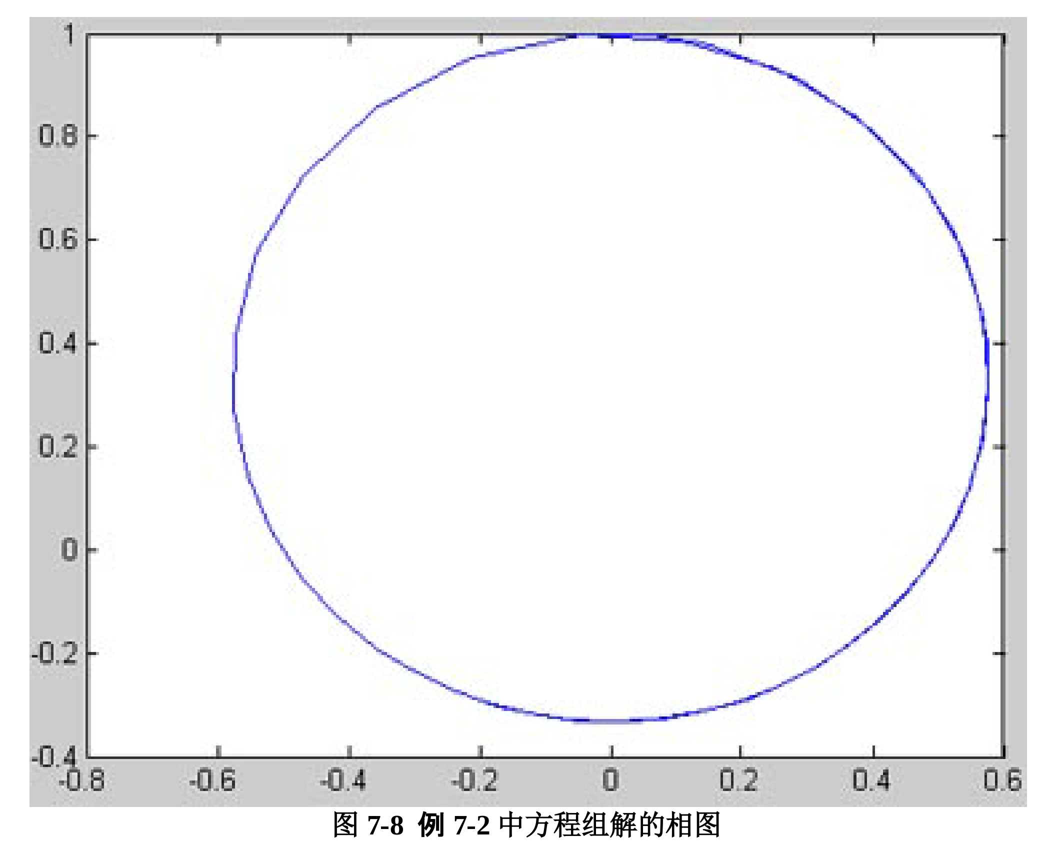


图 7-8 例 7-2 中方程组解的相图

现在我们知道如何求解方程组了，可以使用 *ode45* 求解二阶方程了。

例 7-3

求方程 $y'' + 16y = \sin(4.3t)$ 当 $y(0) = y'(0) = 0$ 时的解。

解 7-3

我们可以把这个方程换成一阶方程组。首先我们设：

$$x_1 = y$$
$$x_2 = y'$$

因此：

$$x_1' = y = x_2$$
$$x_2' = y'' = \sin(4.3t) - 16x_1$$

现在我们创建一个函数实现这个方程组：

```
function xdot = eqx2(t,x);
%创建数组储存数据
xdot = zeros(2,1);
xdot(1) = x(2);
xdot(2) = sin(4.3*t)-16*x(1);
```

我们调用 *ode45* 得到一个解。由于扰动函数是正弦曲线，我们选时间区间为 $0 \leqslant t \leqslant 2\pi$：

```
>> [t,x] = ode45('eqx2',[0 2*pi],[0,0]);
```

现在我们绘制返回的函数图象：

```
>> plot(t,x(:,1),t,x(:,2),'--'),xlabel('t'), axis([0 2*pi -3 3])
```

图象显示在图 7-9 中。

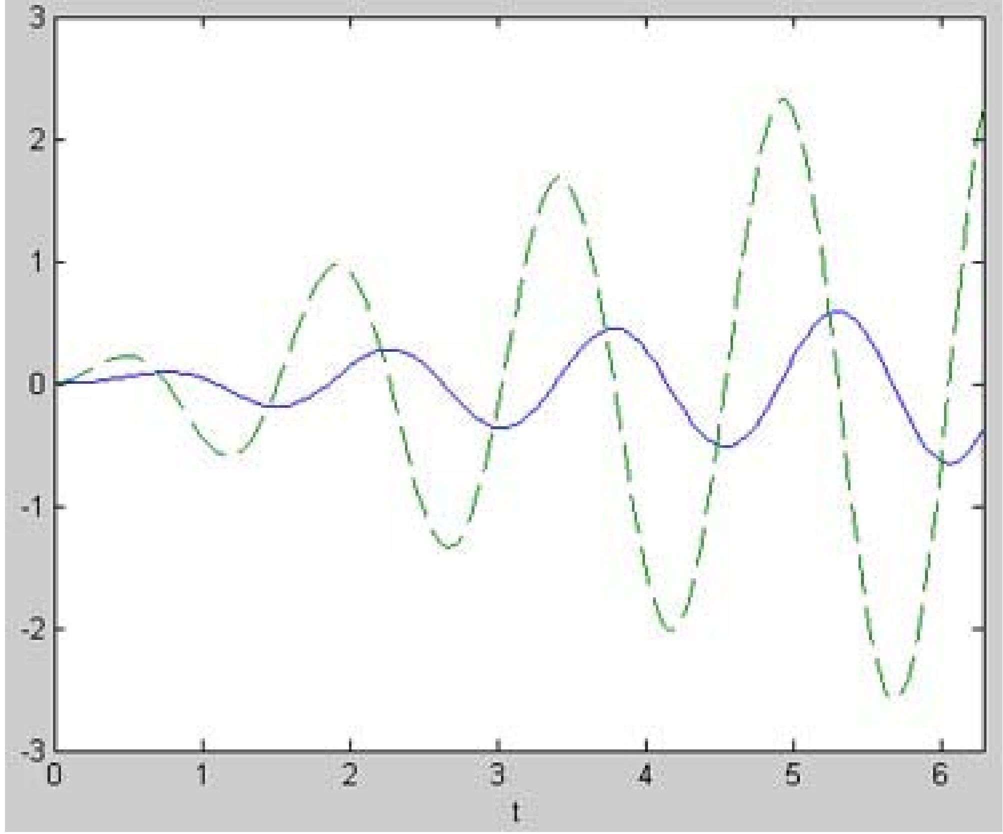


图 7-9 例 7-2 方程组的解，注意振动情况

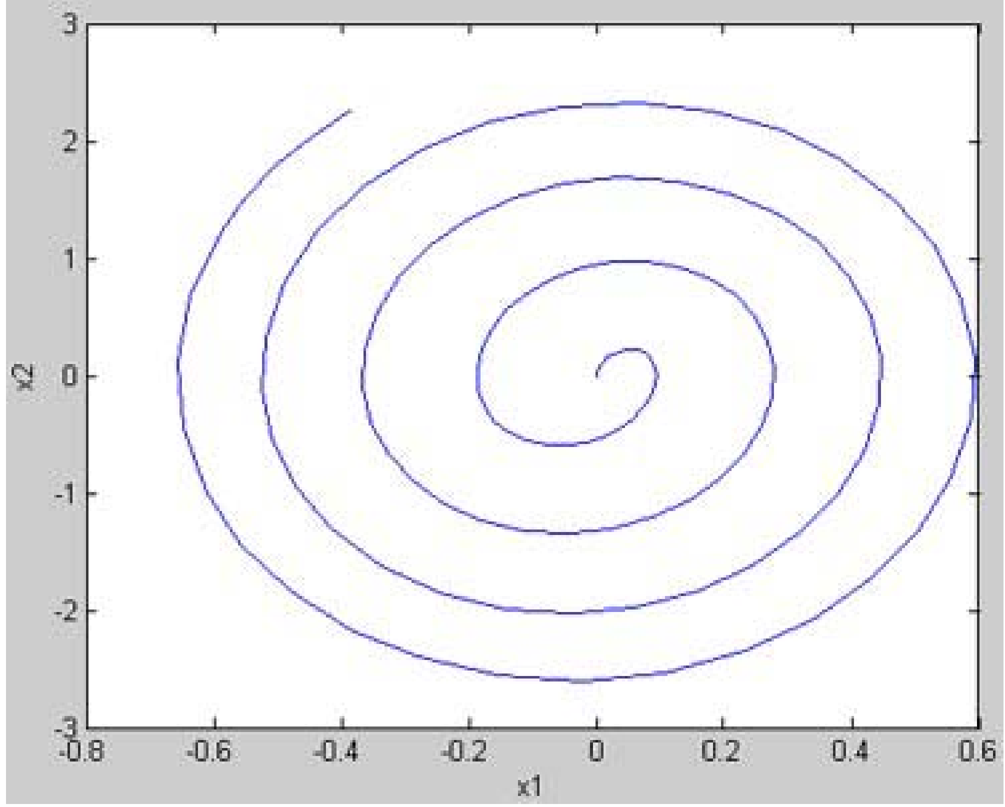


图 7-10 在初始条件 $y(0)=y'(0)=0$ 下例 7-2 中系统的相图

注意解的振幅随着时间增大，而且它们还是异相的，并且第二个解比第一个的振幅要大得多。现在我们产生系统（方程组）的相图：

```
>> plot(x(:,1),x(:,2)),xlabel('x1'),ylabel('x2')
```



结果如图 7-10 所示。

作为比较，考虑题意所给初始条件 $y(0) = y'(0) = 1$ 的情况。让我们重新求解方程并绘制它：

```
>> [t,x] = ode45('eqx2',[0 2*pi],[1,1]);
>> plot(t,x(:,1),t,x(:,2),'--'),xlabel('t'), axis([0 2*pi -4 4])
```

图象如图 7-11 所示。

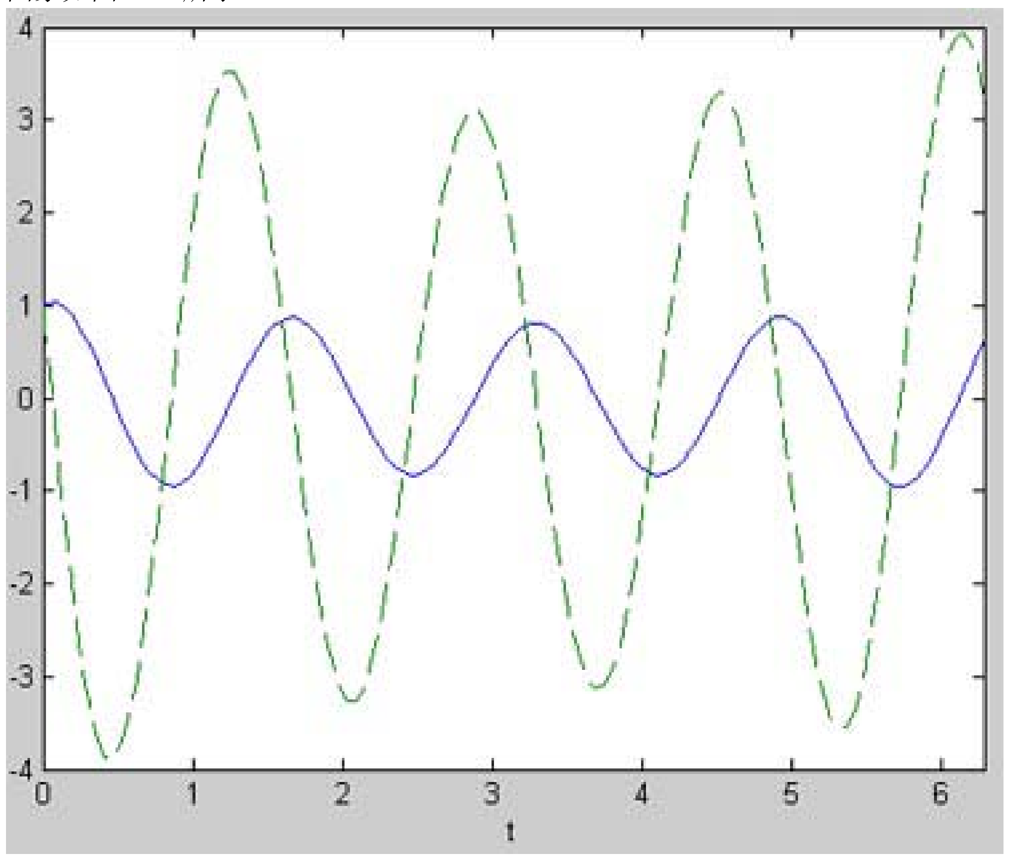


图 7-11 把初始条件设置为 $y(0) = y'(0) = 1$ 后例 7-2 的结果

注意通过改变初始条件后，我们在很大部分上增大了振荡的振幅——函数提前变得更强，x_1 的行为更加有规律。我们再靠近看仔细一点。将 x_1 和 $\cos(4.3t)$ 画在一个图上作比较，如图 7-12。

两个函数的出发点相当接近，不过随着时间的推移渐渐地分开。如果把这种情况与第一种情况即 $y(0) = y'(0) = 0$ 相比较，如图 7-13 所示，它们的相似之处很少，我们很难知道它们有什么关系。

我们在更宽的时间内求解，看看它的振动现象：

```
>> [t,x] = ode45('eqx2',[4*pi 20*pi],[0,0]);
>> plot(t,x(:,1)),xlabel('t')
```

结果如图 7-14，你可以从中看到熟悉的振动情况。

让我们返回 $y(0) = y'(0) = 1$ 的情况。在图 7-15 中，我们绘出与图 7-14 相同时间内的解的图象。仔细看看你会注意到，图 7-14 所示的在 $y(0) = y'(0) = 0$ 情况下的图象中，每次当系统振动到零点附近开始要上升时都会出现反相。这个特征在 $y(0) = y'(0) = 1$ 情况下是没有出现的。

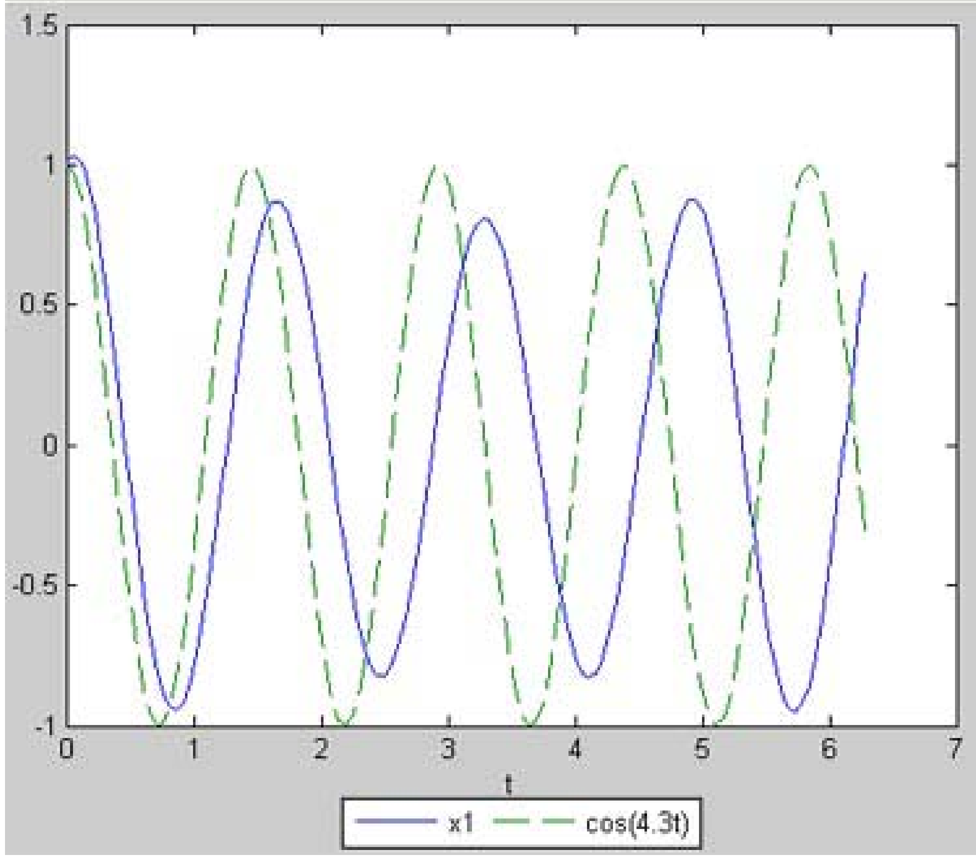


图 7-12 $y(0) = y'(0) = 1$ 时的解 x_1 与 $\cos(4.3t)$的比较

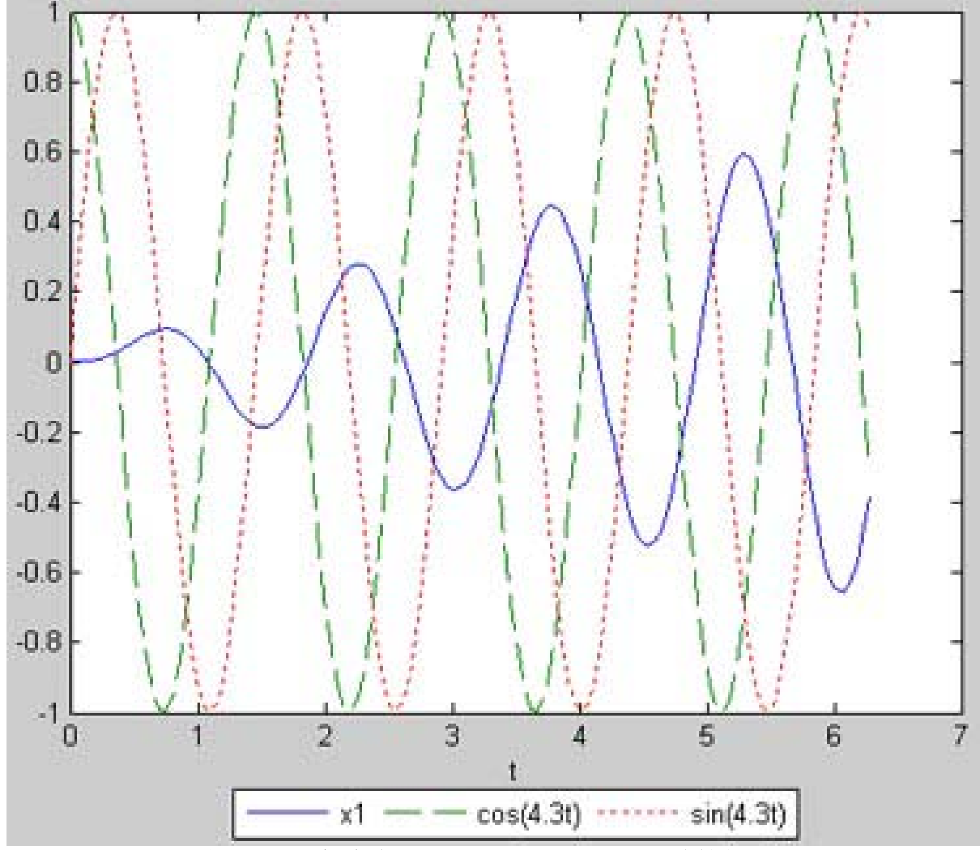


图 7-13 实线表示方程 $y(0) = y'(0) = 0$ 时的解，
它与扰动三角函数或方程的导数并没有多少相似



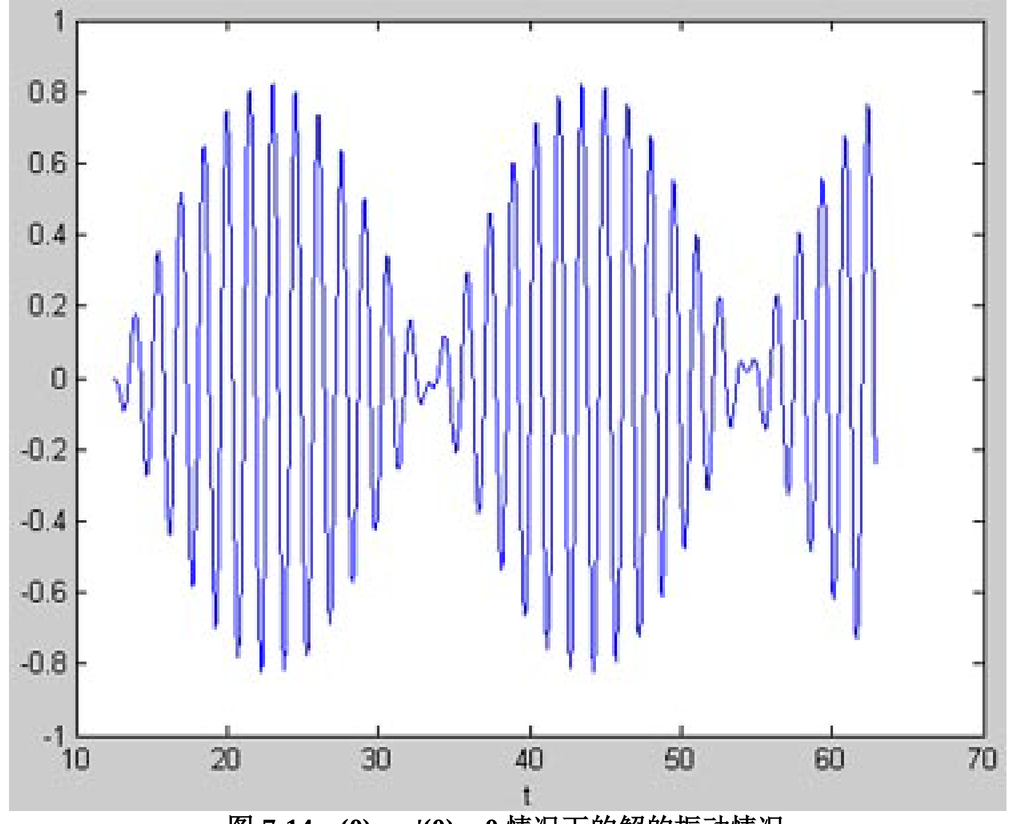




图 7-14 $y(0) = y'(0) = 0$ 情况下的解的振动情况

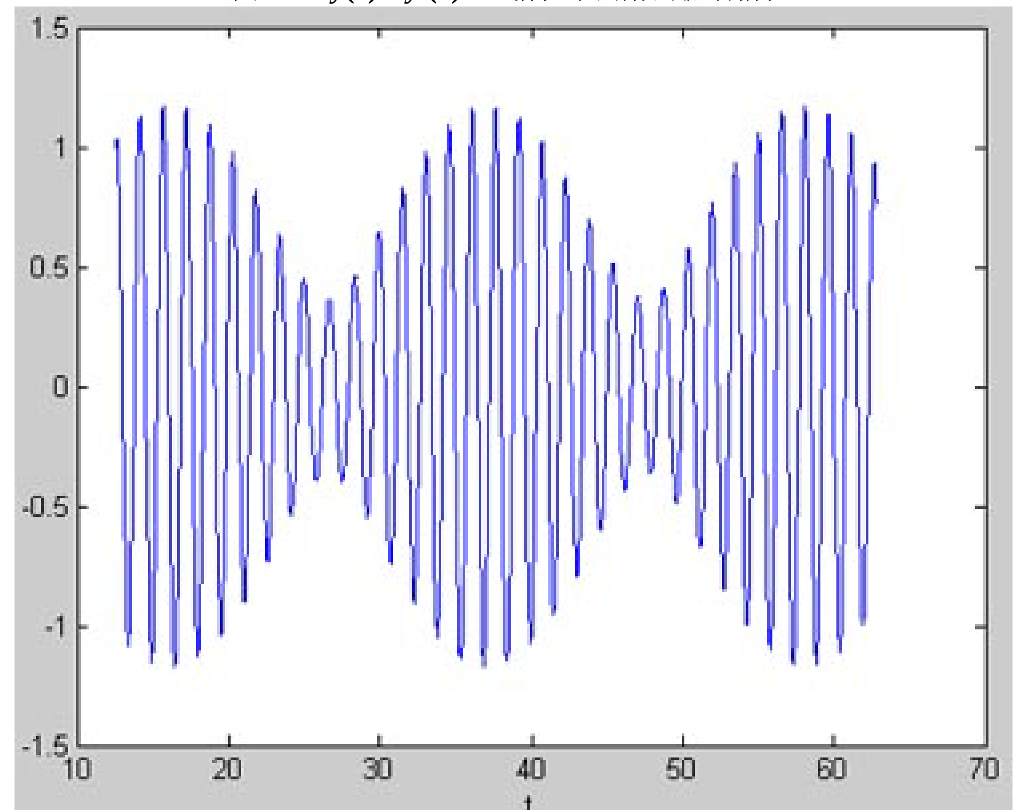


图 7-15 当 $y(0) = y'(0) = 1$ 时，解的振幅先缩小再变大——不过没有出现反相的情况

最后，我们产生 `y(0) = y'(0) = 1` 的相图，如图 7-16 所示。注意当它与图 7-10 相比较时的不同。它告诉我们关于方程的解的什么内容了？

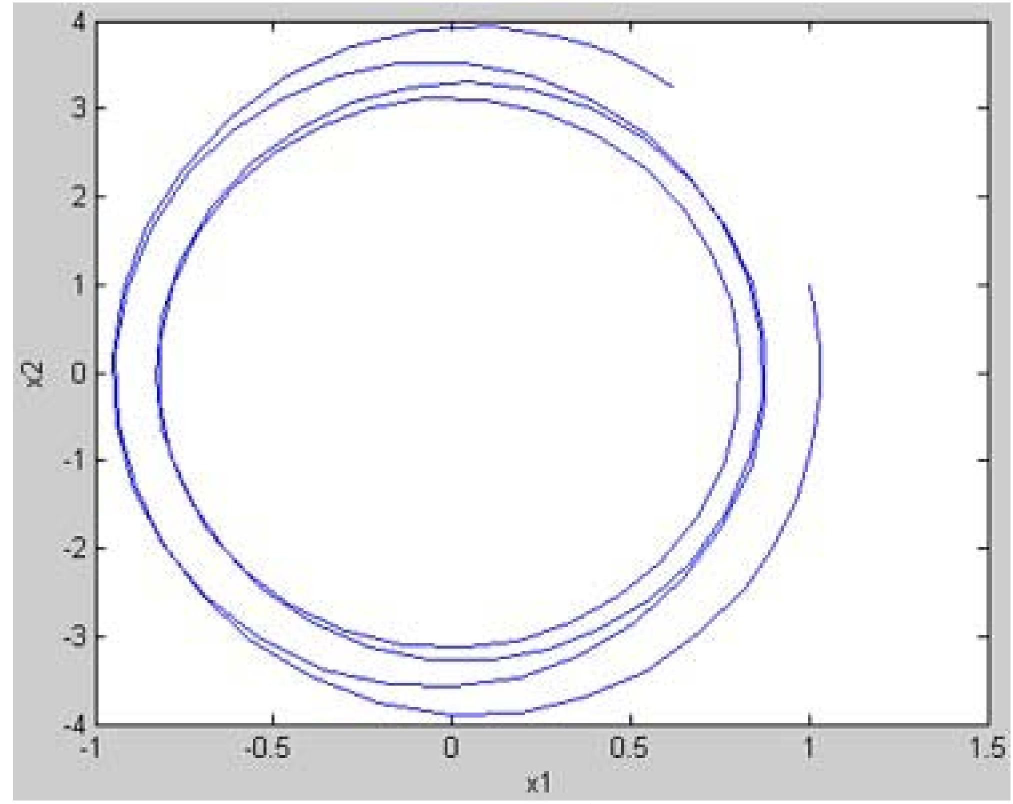


图 7-16 例 7-2 中方程组的相图，条件取 $y(0) = y'(0) = 1$

习题

习题

1. 求下列方程组的解：

$$x_1' = 2x_1x_2$$
$$x_2' = -x_1^2$$
$$x_1(0) = x_2(0) = 0$$

2. 求下面方程组的解：

$$x_1' = x_2$$
$$x_2' = -x_1$$
$$x_1(0) = x_2(0) = 1$$

3. 解方程：

$$y' = -2.3y,\ y(0) = 2$$

（原书 y(0)＝0.根据答案，应该是 y(0)=2 才对。）

4. 分别使用 *ode23* 和 *ode45* 求解下列方程：

$$y' = y,\ y(0) = 1$$

ode23 和 *ode45* 分别返回多少个点？

5. 解方程 $y' = -ty + 1,\ \ y(0) = 1$。
6. 解方程 $y' = t^2y,\ \ y(0) = 1$ 取 $0 \le t \le 2$。
7. *ode23* 和 *ode45* 使用的数值技术分别是什么？
8. 解方程 $\frac{dy}{dt} = \frac{2}{\sqrt{1 - t^2}}$，$-1< t <1$，$y(0) = 1$。（本题应该是不能用 ode 函数求解，求答案。如果你有求解的方法，请及时告诉我。）
9. 解方程 $y'' - 2y' + y = \exp(-t)$，$y(0) = 2$， $y'(0) = 0$。
10. 绘制上面第 9 题方程组的相图。

【参考答案在第 234 页】

第八章



积分

本章介绍了计算积分的各种方法。我们从求解符号积分开始，然后再进一步学习数值积分。

INT 命令

设 *f* 是 **MATLAB** 中表示的一个表达式。我们使用下面的格式就可以得到 *f* 的一个积分表达式：

```
int(f)
```

表达式 *f* 在输入时可以先创建一个新变量或先进行引用，或者直接用单引号引起来传递给 *int*。例如，我们演示下面的情况：

$$\int x dx = \frac{1}{2}x^2$$

(省略积分常量)写成：

```
>> int('x')
```

MATLAB 返回：

```
ans =
1/2*x^2
```

MATLAB 解得的全是符号形式的积分，另一个简单例子：

```
>> int('x^2')
ans =
1/3*x^3
```

而更一般的例子是：

```
>> int('x^n')
ans =
x^(n+1)/(n+1)
```

如果我们什么也不说，*int* 自己猜测对哪个变量进行积分。例如，我们定义一个三角函数：

```
>> g = 'sin(n*t)';
```

如果我们把这个函数传递给 *int*，它将假设 *t* 就是积分变量：

```
>> int(g)
ans =
-1/n*cos(n*t)
```

当然，我们也可以使用 *int(f, v)* 语法来调用 *int*，其中 *f* 就是要积分的函数，而 *v* 是积分变量。仍然使用 *g*，我们写成：

```
>> syms n
>> int(g,n)
ans =
-1/t*cos(n*t)
```

对于计算积分的读者来说，不要忘记在你的答案后面带上积分常量。当创建符号表达式



时，并不总是需要把它们放在括号内——记得要先使用 *syms* 命令。如果我们输入：

```
>> g = b^t;
```

MATLAB 会抱怨说：

```
??? Undefined function or variable 'b'.
```

用下面的方法我们就能绕过去。首先我们调用 *syms* 告诉 **MATLAB** 我们要使用哪个符号作为变量，然后我们定义函数，但没有把它用单引号引起来：

```
>> syms a t
>> g = a*cos(pi*t)
g =
a*cos(pi*t)
>> int(g)
ans =
a/pi*sin(pi*t)
```

例 8-1

$f(x) = b^x$ 的积分是什么？取 $b = 2$、$x = 4$ 计算解的值。

解 8-1

一开始我们定义符号变量：

```
>> syms b x
```

现在我们定义函数并计算它的积分：

```
>> f = b^x;
>> F = int(f)
F =
1/log(b)*b^x
```

通过调用 *subs* 命令，使用给定的条件我们可以计算表达式的数值。要在一个命令中用数代替符号变量，我们要把被代替的符号列表和用来代替的数值列表都用花括号括起来，在本例中我们写成：

```
>> subs(F,{b,x},{2,4})
ans =
   23.0831
```

例 8-2

计算 $\int x^5\cos(9x)dx$ 。

解 8-2

手工计算这个积分需要用到分部积分法，而且还很伤脑筋。如果用 **MATLAB**，我们用一行就可以求得答案：

```
>> F = int(x^5*cos(x))
F =
x^5*sin(x)+5*x^4*cos(x)-20*x^3*sin(x)-60*x^2*cos(x)+120*cos(x)+120*x*sin(x)
```

我们可以使用“*pretty*”命令，让 **MATLAB** 以比较适合人阅读的格式显示答案：

```
>> pretty(F)
  x^5sin(x) + 5 x^4cos(x) - 20 x^3sin(x) - 60 x^2cos(x) + 120 cos(x) + 120 x sin(x)
```

例 8-3

求$\int 3y\ \sec(x)dy$ 。

解 8-3

被积函数含有两个变量，因此我们告诉 **MATLAB** 要对 y 进行积分：

```
>> syms y
>> int('3*y^2*sec(x)',y)
ans =
y^3*sec(x)
```

如果我们想对 x 进行积分，我们将写成：

```
>> syms x
>> int('3*y^2*sec(x)',x)
ans =
3*y^2*log(sec(x)+tan(x))
```

定积分

int 还可以用来计算定积分，此时要把积分区间传递给它。如果我们输入 *int(f, a, b)*，那么 **MATLAB** 将计算默认独立变量积分然后返回：

$$\int_a^b f(x)dx = F(b) - F(a)$$

例如：

$$\int_2^3 xdx = \frac{1}{2}x^2\Big|_2^3 = \frac{1}{2}(9 - 4) = \frac{5}{2}$$

要在 **MATLAB** 中计算的话将写成：

```
>> syms x
>> int('x',2,3)
ans =
5/2
```

相同地，如果我们要 **MATLAB** 产生中间表达式$\frac{1}{2}x^2$ ，我们将写成：

```
>> syms x
>> F = int('x')
F =
1/2*x^2
>> a = subs(F,x,3) - subs(F,x,2)
a =
    2.5000
```

例 8-4

曲线 $f(x) = x^2\cos x$ 下面在$-6 \leqslant x \leqslant 6$ 范围内的面积是多少？

解 8-4



这条曲线如图 8-1 所示。要求得曲线下面的面积，即要求我们计算：

$$\int_{-6}^{6} x^2 \cos x dx$$

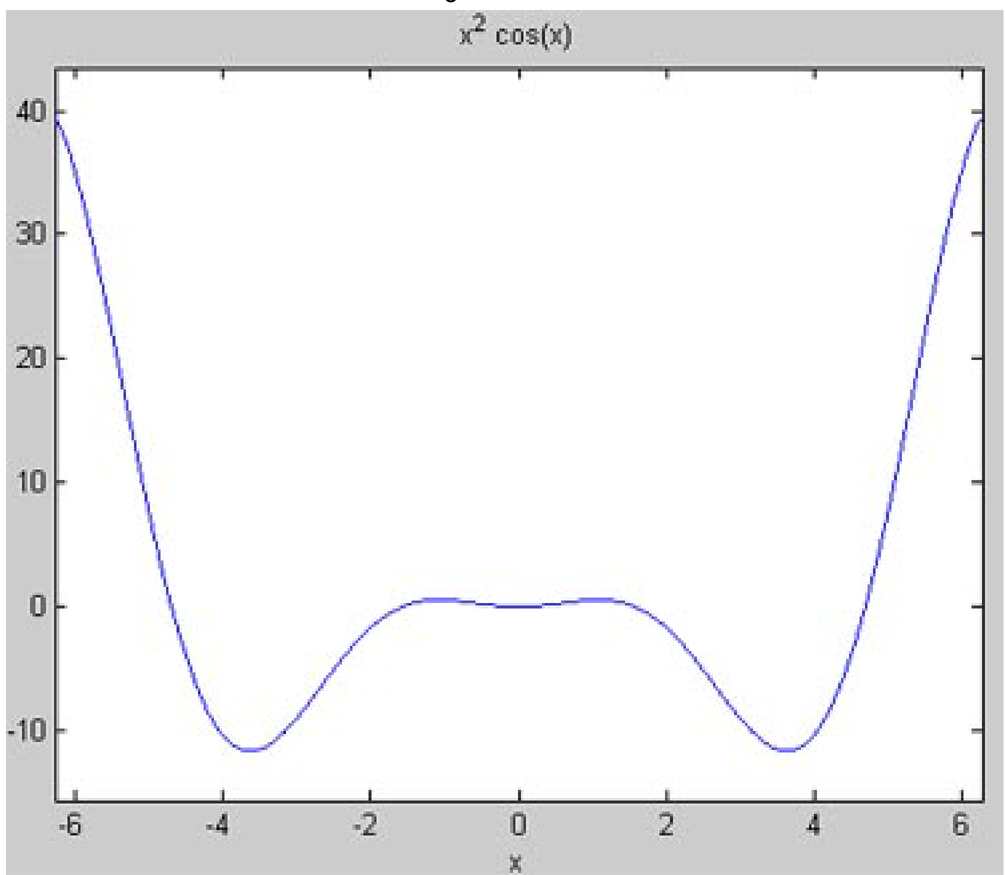


图 8-1 $-6 \le x \le 6$ 区间内的 $f(x) = x^2 \cos x$ 图象

首先定义函数：

```
>> syms x
>> f = x^2*cos(x);
```

现在我们求它的积分：

```
>> a = int(f,-6,6)
a =
68*sin(6)+24*cos(6)
```

要得到数值结果，我们使用 *double* 计算一遍：

```
>> double(a)
ans =
    4.0438
```

例 8-5

计算 $\int_0^{\infty} e^{-x^2} \sin x dx$

解 8-5

告诉 **MATLAB** 我们要计算无穷区间内的积分，积分上限我们使用 *inf* 表示：

```
>> syms x
>> a = int(exp(-x^2)*sin(x),0,inf)
a =
-1/2*i*pi^(1/2)*erf(1/2*i)*exp(-1/4)
```

现在我们计算结果的数值：

```
>> double(a)
ans =
    0.4244
```

例 8-6

求曲线 e^{-x} 关于 x 轴旋转得到的旋转体在 $1 \leq x \leq 2$ 内的体积。

解 8-6

曲线如图 8-2 所示。曲线 $f(x)$ 关于 x 轴旋转得到的旋转体体积由下式得出：

$$\int_a^b \pi[f(x)]^2 dx$$

本例中的被积函数是：

$$\pi(e^{-x})^2 = \pi e^{-2x}$$

因此这个立体的体积是：

```
>> syms x
>> int(pi*exp(-2*x),1,2)
ans =
-1/2*pi*exp(-4)+1/2*pi*exp(-2)
```

这个结果的数值是：

```
>> double(ans)
ans =
    0.1838
```

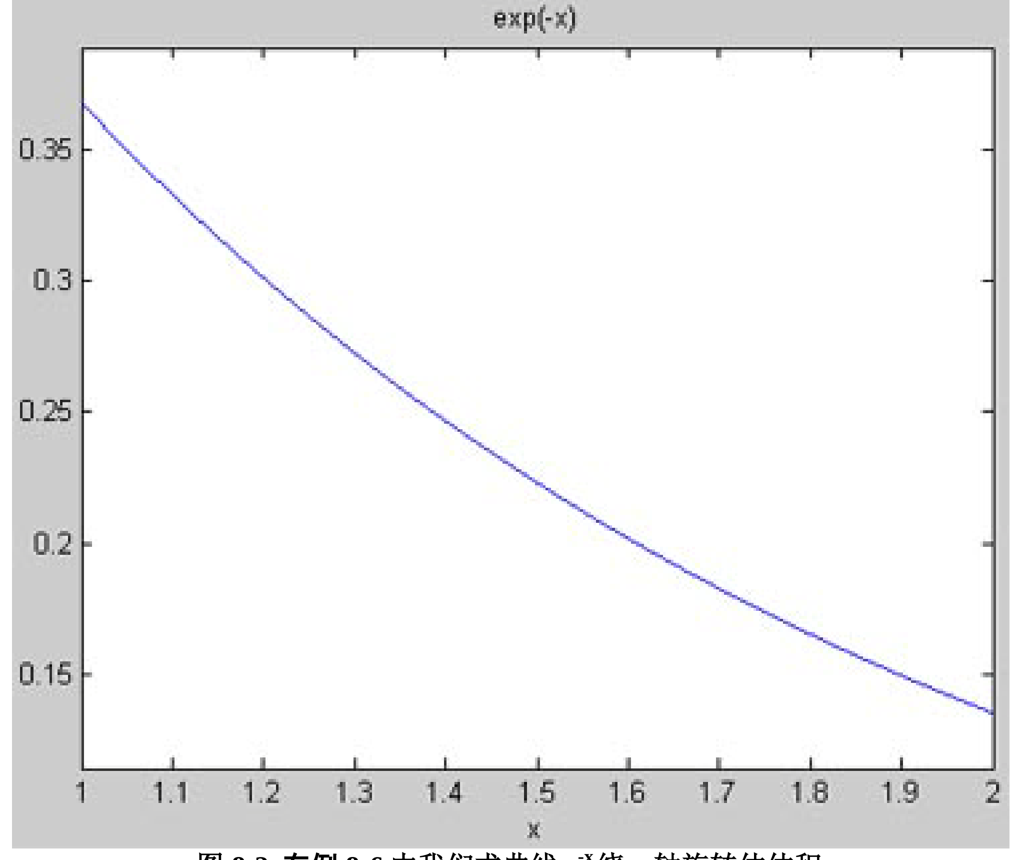


图 8-2 在例 8-6 中我们求曲线 e^{-x} 绕 x 轴旋转体体积

例 8-7

sinc 函数在有限带宽信号处理中非常有用，它的定义是：

$$f(x) = \frac{\sin(x)}{x}$$

求 *sinc* 函数及 *sinc* 的平方函数分别在 $-20 \leqslant x \leqslant 20$ 和 $-\infty < x < \infty$ 上的积分。

解 8-7

让我们定义这两个函数：

```
>> syms x;
>> sinc = sin(x)/x;
>> sinc_squared = sinc^2;
```

sinc-squared 函数在描述光束传输方面有应用。首先我们绘制这两个函数，第一个是 *sinc* 函数：

```
>> ezplot(sinc,[-20 20])
>> axis([-20 20 -0.5 1.2])
```

sinc 函数的图象如图 8-3。

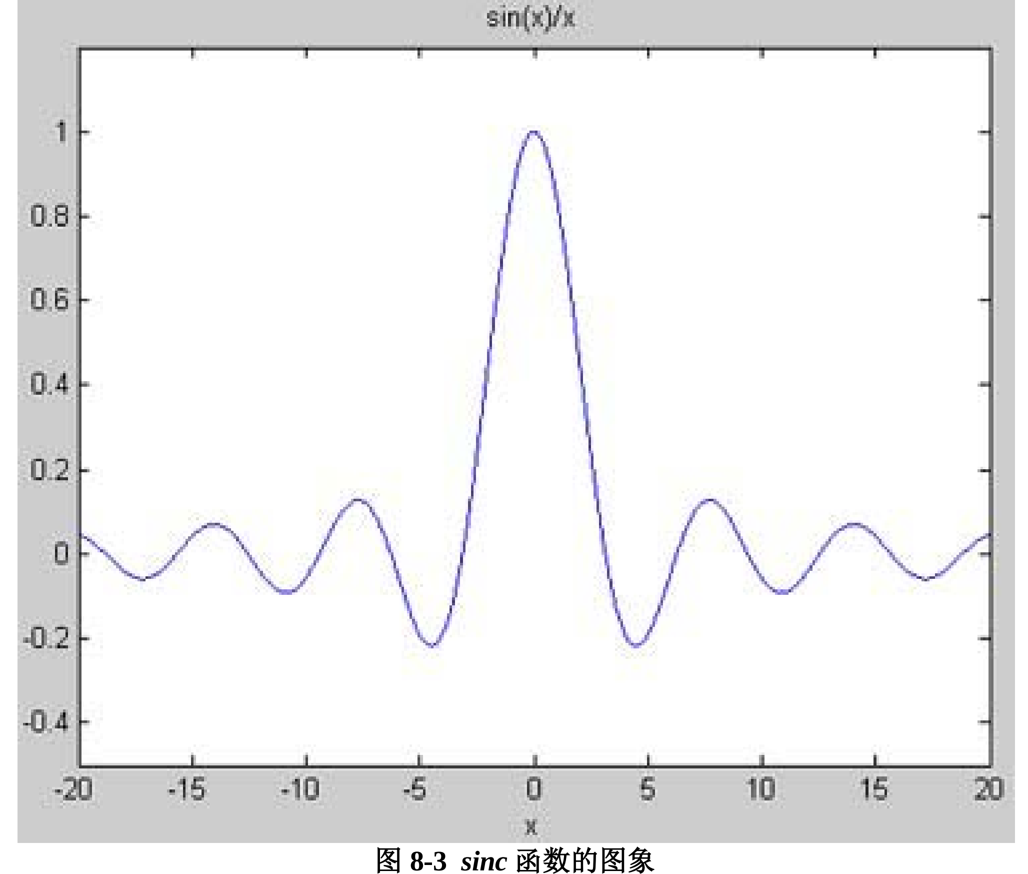


图 8-3 *sinc* 函数的图象

现在我们绘制它的平方图象：

```
>> ezplot(sinc_squared,[-10 10])
>> axis([-10 10 -0.1 1.2])
```

sinc-squared 函数的图象如图 8-4 所示。

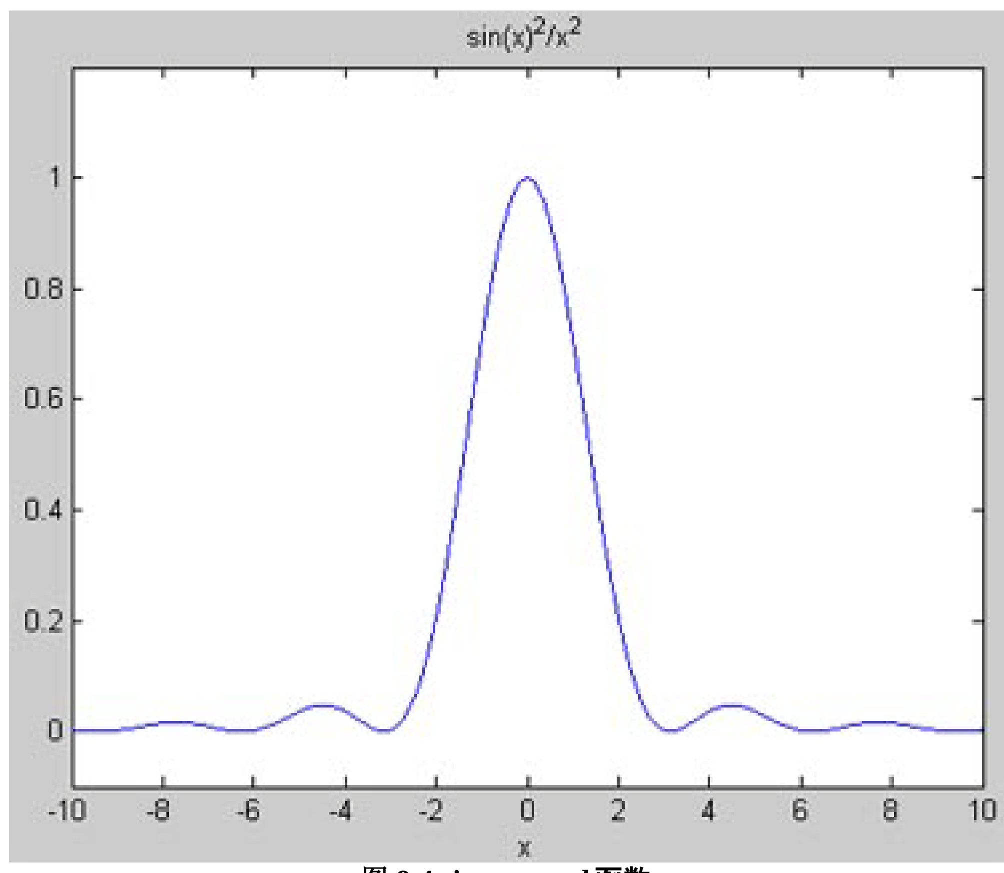


图 8-4 *sinc-squared* 函数

现在我们计算-20≤x≤20 之间的积分。对于 *sinc* 函数，我们求得：

```
>> a = int(sinc,-20,20)
a =
2*sinint(20)
```

我们可以使用 *double* 计算它的数值：

```
>> double(a)
ans =
    3.0965
```

现在我们对 *sinc-squared* 求积分：

```
>> b = int(sinc_squared,-20,20)
b =
-1/20+1/20*cos(40)+2*sinint(40)
```

数值是：

```
>> double(b)
ans =
    3.0906
```

这两个结果非常接近。$-\infty < x < \infty$间的积分是：

```
>> a = int(sinc,-inf,inf)
a =
```

```
pi

>> b = int(sinc_squared,-inf,inf)
b =
pi
```

实际上，在$-a \le x \le a$范围内，我们发现当 a 越大这两个函数越趋近 π。*sinc-squared* 函数会趋近得快一些，这是由于 *sinc* 函数的圆形突出部份会在正负之间交替变换，因此会相互抵消掉一些：

```
>> a = double(int(sinc,-50,50))
a =
    3.1032
>> b = double(int(sinc_squared,-50,50))
b =
    3.1217
```

在整个曲线上面的部分，它们之间的差异会越来越少，因此两个函数覆盖相同的面积。

多重积分

通过嵌套使用 *int* 语句我们能够在 **MATLAB** 中计算多重积分。假设我们要计算不定积分：

$$\int\int\int xy^2z^5dxdydz$$

使用下面的方法即可做到：

```
>> syms x y z
>> int(int(int(x*y^2*z^5,x),y),z)
ans =
1/36*x^2*y^3*z^6
```

定积分的处理类似。我们计算：

$$\int_2^1\int_4^2 x^2ydxdy$$

使用的命令是：

```
>> f = x^2*y;
>> int(int(f,x,2,4),y,1,2)
ans =
28
```

当计算圆柱坐标和球面坐标中的多重积分时，记得正确输入面积和体积元素——**MATLAB** 不会自动做。

例 8-8

求高为 h 半径为 a 的圆柱体体积。$a = 3.5$ 英寸和 $h = 5$ 英寸时圆柱体的体积为多少?

解 8-8

我们使用圆柱坐标进行积分：

$$0 \le r \le a,\ 0 \le \theta \le 2\pi,\ 0 \le z \le h$$

圆柱坐标中的体积元素是：

$$dV = r\ dr\ d\theta\ dz$$

所以高为 h 半径为 a 的圆柱体的体积为：

$$V = \int_h^0 \int_{2\pi}^0 \int_a^0 r dr d\theta dz$$

在 **MATLAB** 中实现它的命令是：

```
>> syms r theta z h a
>> V = int(int(int(r,r,0,a),theta,0,2*pi),z,0,h)
V =
a^2*pi*h
```

半径 $a = 3.5$ 英寸和高度 $h = 5$ 英寸的圆柱体体积是：

```
>> subs(V,{a,h},{3.5,5})
ans =
  192.4226
```

答案是以立方英寸为单位的。

数值积分

通过调用 *trapz*(*x*, *y*)函数 **MATLAB** 可以计算梯形积分。这里 x 和 y 是两个数组，x 是积分的定义域，而 y 是在那些点上取得的函数值。可以对多个函数同时进行积分（在同一个定义域 x 上），只需用多列的形式把每个函数的 y 值传递过去。

你可能已经很熟悉微积分学上的梯形法，即把某条曲线下的区域分成一系列的矩形。然后把每个矩形的面积累加起来得到积分。

例 8-9

计算

$$\int_0^2 x^2 dx$$

把区域分成 10 等分和 20 等分，然后分别计算每种情况的相对误差。

解 8-9

首先我们用解析法求这个问题的精确答案：

$$\int_0^2 x^2 dx = \frac{1}{3}x^3 \Big|_2^0 = \frac{8}{3} = 2.667$$

现在我们把区域分成 10 等分：

```
>> x = linspace(0,2,10);
```

定义函数：

```
>> y = x.^2;
```

进行积分，求得：

```
>> a = trapz(x,y)
a =
    2.6831
```

相对误差是：

```
>> 100*abs((8/3-a)/(8/3))
ans =
    0.6173
```

现在我们用 20 等分重复这个过程，看看如何减少误差：

```
>> x = linspace(0,2,20);
>> y = x.^2;
>> a = trapz(x,y)
a =
    2.6704
>> 100*abs((8/3-a)/(8/3))
ans =
    0.1385
```

可以看到，把矩形的数量加倍后，误差差不多减少了 5 倍。

例 8-10

数值估算下面高斯函数的积分：

$$\int_{-2}^{2} e^{-x^2} dx \quad 和 \quad \int_{-\infty}^{\infty} e^{-x^2} dx$$

计算相对误差。把第一个例子分成 200 等分。

解 8-10

我们用 *long* 格式显示结果：

```
>> format long
```

对于第一个式子我们求得：

```
>> x = linspace(-2,2,200);
>> gauss = exp(-x.^2);
>> in = trapz(x,gauss)
in =
   1.76415784847621
```

这个积分的精确值是：

$$\int_{-2}^{2} e^{-x^2} dx = \sqrt{\pi}\mathrm{Erf}(2)$$

其中 Erf(2)是**误差函数**。**MATLAB** 计算的结果是：

```
>> ac = sqrt(pi)*erf(2)
ac =
   1.76416278152484
```

我们看到结果在小数点后四位是一致的。以百分比显示的相对误差是：

```
>> d = ac - in;
>> err = abs(d/ac)*100
err =
    2.796254794950795e-004
```

对于第二个式子的积分，解析解的结果是：

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

让我们把高斯图象绘制出来，思考如何做数值积分，如图 8-5。

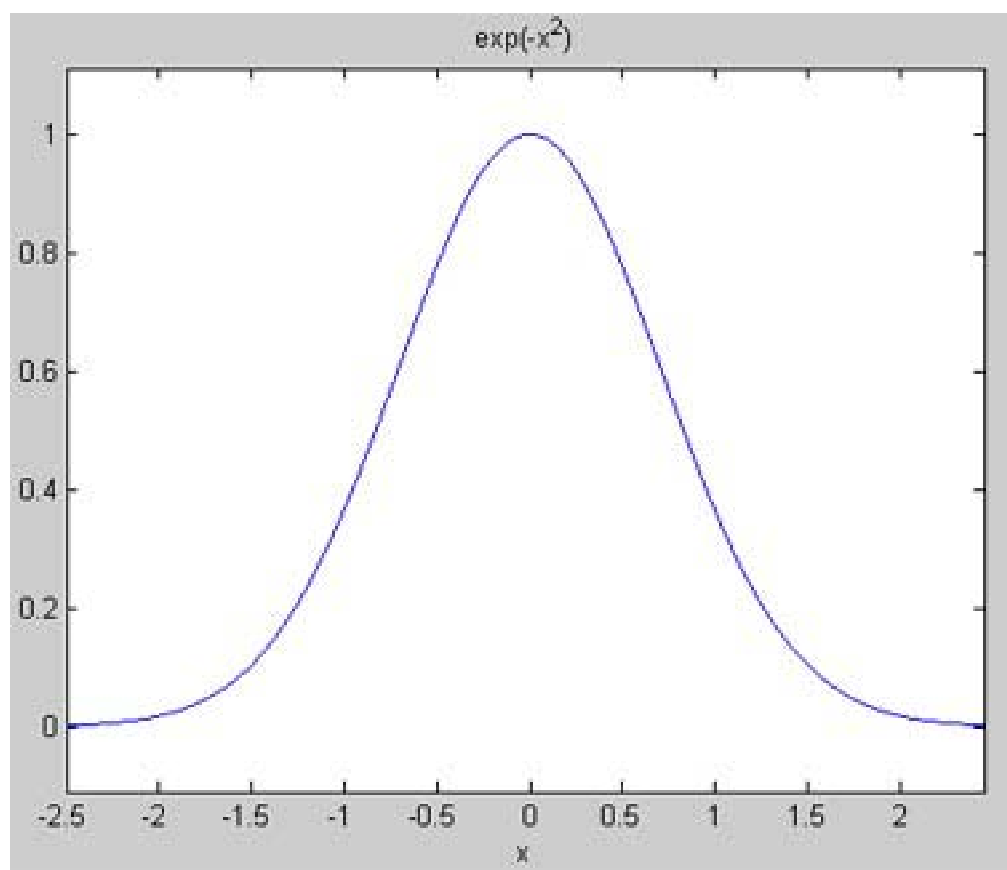


图 8-5 以原点为中心的高斯函数图象

一般地，高斯函数可以写成下面的形式：

$$\varphi(x) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

其中 μ 是均值而 σ 是标准差。结果表明 99.73% 的面积落在 3 倍标准差范围之内。在本例中是 $exp(-x^2)$。以原点为中心的高斯函数（因此 $\mu=0$）有：

$$\frac{1}{2\sigma} = 1, \Rightarrow \sigma = \frac{1}{\sqrt{2}}$$

这意味着在我们这个例子中，99.73%的面积落在曲线区间：

$$x = \pm\frac{3}{\sqrt{2}} \approx \pm 2.213$$

因此我们可以尝试在 $-2.2 \leqslant x \leqslant 2.2$ 上进行数值积分。让我们把这个范围分成 200 等分：

```
>> x = linspace(-2.2,2.2,200);
>> gauss = exp(-x.^2);
```

求积分后我们得到：

```
>> est = trapz(x,gauss)
est =
   1.76914920736586
```

实际的解析值是：

```
>> ac = sqrt(pi)
ac =
   1.77245385090552
```

数值积分

相对误差是：

```
>> d = ac - est;
>> err = abs(d/ac)*100
err =
   0.18644454624110
```

与整个范围内解析得到的结果相比，0.18%也算不上是个很糟糕的误差。让我们把范围扩伸到-3≤x≤3 进行积分：

```
>> x = linspace(-3,3,200);
>> gauss = exp(-x.^2);
>> est = trapz(x,gauss)
est =
   1.77241458438211
```

你可以看到，新的结果已经在精度上有了提升，现在的相对误差是：

```
>> d = ac - est;
>> err = abs(d/ac)*100
err =
   0.00221537634860
```

0.002 %的相对误差表示现在的精度又提高了。

例 8-11

轨道上的火箭车的速度每秒测量一次，共 10 次，以 ft/s 为单位结果如下：

t	1	2	3	4	5	6	7	8	9	10
$v(t)$	65	95	110	150	170	210	240	260	265	272

求火箭通过的总距离。

解 8-11

速度与位置的关系是：

$$v = \frac{dx}{dt}$$

因此我们对速度进行积分就可以求得位移：

$$\int_a^b v(t)dx$$

要进行数值积分，首先我们要把数据放在两个数组中：

```
>> t = [1:1:10];
>> v = [65 95 110 150 170 210 240 260 265 272];
```

我们可以使用下面的公式计算每个元素的值：

$$x(t_{k+1}) = \int_{t_k}^{t_{k+1}} v(t)dt + x(t_k)$$

我们把第一个元素初始化为零：

```
>> x(1) = 0;
```

现在我们使用 *for* 循环进行积分：

```
>> for k = [1:9]
x(k+1) = trapz(t(k:k+1),v(k:k+1))+x(k);
end
```

结果是：

时间	位移（单位：1000 英尺）
1	0
2	0.08000000000000
3	0.18250000000000
4	0.31250000000000
5	0.47250000000000
6	0.66250000000000
7	0.88750000000000
8	1.13750000000000
9	1.40000000000000
10	1.66850000000000

因此火箭车运动了 1668.50 英尺，上面的技巧给了我们每秒的位置。如果只是想知道末位置，调用一次 *trapz* 函数就可以得到结果了：

```
>> trapz(t,v)
ans =
    1.668500000000000e+003
```

正交积分

MATLAB 有两个命令 *quad* 和 *quad1* 可以用来实现正交积分。这种类型的方法基于使用二次函数代替矩形更能接近曲线下方面积的原理（使用高阶的多项式还能够得到更精确的结果）。辛普森法则（*Simpson's rule*）是把积分区间分成偶数段，相邻两段下面的面积用不同的二次函数近似表示。*quad* 函数采用适应辛普森法则的逼近方法进行数值积分。要使用 *quad*，把被积的函数传递给它，后面跟着积分区间。

quadl 函数采用洛巴托（*Lobatto*）积分法（注意 *quandl* 最后面的“*l*”是英文字母 *L* 的小写形式）。这是一种适合更加灵活复杂的类型，查阅 **MATLAB** 的帮助以便获得更多的信息。

quad 和 *quadl* 函数的不足之处是无法对点集进行积分。

例 8-12

使用正交积分计算

$$\int_0^{1/8} e^{-x^2} dx$$

解 8-12

小数点后 4 位的解析值是：

$$\frac{1}{2}-\frac{1}{2e^{1/4}} \approx 0.1106$$

quad 函数返回的是：

```
>> quad('exp(-2*x)',0,1/8)
ans =
    0.1106
```

事实上如果我们计算到小数点后 14 位，我们会发现它们仍然极其一致：

```
>> format long
>> ac = 1/2 - 1/(2*exp(1/4))
ac =
   0.11059960846430
```



```
>> quad('exp(-2*x)',0,1/8)
ans =
   0.11059960846436
```

使用 *trapz* 函数我们也能得到不错的答案，不过在第 7 位之后开始不同：

```
>> x = linspace(0,1/8,100);
>> y = exp(-2*x);
>> r = trapz(x,y)
r =
   0.11059966723785
```

习题

1. 计算$\int xe^{-ax}dx$

2. 计算$\int \frac{dx}{\sqrt{x^2+bx+c}}$

3. 计算$\int x\tan x dx$

4. 求$\int \sin^2(yx)dy$

5. 计算$\int_0^\infty e^{-3x}\cos x dx$

6. 求曲线$\sqrt{x}$和 x^2 在 $0\leqslant x\leqslant 1$ 内所包含的区域旋转一周所围成的体积。

7. 使用 **MATLAB** 产生半径为 a 的球体表面积公式和体积公式，并计算半径 $a = 2\text{m}$ 时的体积。

8. 使用梯形积分法计算数值积分$\int_0^\pi \cos(\pi x)dx$，把函数分成 50 等分，并计算相对误差。

9. 计算高斯函数在$-2.2\leqslant x\leqslant 2.2$ 上的数值积分，把函数分成 1000 等分。看看是不是分得越多相对误差就减少？

10. 数值计算：

$$\int_0^{10} J_1(x)dx$$

其中 $J_1(x)$是使用梯形和正交方法的第一类贝塞耳函数（*Bessel function*）。

【参考答案在第 239 页】

第九章



变换

变换，比如拉普拉斯（*Laplace*）变换、z 变换和傅立叶（*Fourier*）变换，在科学和工程上被广泛应用。变换除了能够简化分析，它还能够让我们以新的观点看待数据。例如，傅立叶变换能够让我们把一个用时间函数表示的信号看成是频率函数的信号。

在本章中我们将向读者介绍使用 **MATLAB** 处理变换的基础知识，我们会介绍 *laplace*、*fourier* 和 *fft* 命令。

拉普拉斯变换

时间函数 $f(t)$的**拉普拉斯变换**用下面的积分式定义：

$$\mathscr{L}\{f(t)\} = \int_0^\infty f(t)e^{-st}dt$$

我们通常把$f(t)$的拉普拉斯变换写$F(s)$。拉普拉斯变换的优点是能把微分方程变成代数方程。从原理上来说，求解代数方程要容易得多，但实际上并不总是这样。但是使用**MATLAB**求解的情况会大大简化。

在**MATLAB**计算拉普拉斯变换，我们要调用$laplace(f(t))$，它做的是符号计算。让我们使用这个函数建立一系列你可能会在微分方程和电子工程书上遇到的函数的拉普拉斯变换。

我们从最简单的例子开始，计算常数方程$f(t)=a$的拉普拉斯变换。首先我们定义符号变量：

```
>> syms s t
```

首先，如果我们计算数值1的拉普拉斯变换，看看会发生什么情况：

```
>> laplace(1)
??? Function 'laplace' is not defined for values of class 'double'.
```

因此我们需要先定义符号常量：

```
>> syms a
```

现在我们常量a的拉普拉斯变换：

```
>> laplace(a)
ans =
1/s^2
```

下面是一些t的高次幂的拉普拉斯变换：

```
>> laplace(t^2)
ans =
2/s^3

>> laplace(t^7)
ans =
5040/s^8

>> laplace(t^5)
ans =
120/s^6
```

从这些我们演绎出有名的公式：

$$\mathscr{L}(t^n) = \frac{n!}{s^{n+1}}$$

我们继续看一些在科学和工程上常用的函数的拉普拉斯变换。首先我们考虑指数衰减函数：

```
>> syms b
>> laplace(exp(-b*t))
ans =
1/(s+b)
```

sin和cos函数的拉普拉斯变换是：

```
>> syms w
>> laplace(cos(w*t))
ans =
s/(s^2+w^2)

>> laplace(sin(w*t))
ans =
w/(s^2+w^2)
```

我们计算双曲余弦函数的拉普拉斯变换：

```
>> laplace(cosh(b*t))
ans =
s/(s^2-b^2)
```

拉普拉斯变换是线性的，即：

$$\mathcal{L}\{af(t) + bg(t)\} = a\mathcal{L}\{f(t)\} + b\mathcal{L}\{g(t)\}$$

我们使用**MATLAB**验证。设

```
>> f = 5 + exp(-2*t);
```

求得它的拉普拉斯变换是：

```
>> laplace(f)
ans =
5/s+1/(s+2)
```


拉普拉斯逆变换

读者在学习电路分析课程时计算拉普拉斯逆变换是件非常头痛的事，计算的过程枯燥无味，并且含有很多很难的代数内容。不过如果使用**MATLAB**，我们的难题迎刃而解。要计算拉普拉斯逆变换，我们输入*ilaplace*，不用我们再去做部分分式展开和其它一些讨厌的事。首先我们使用一些简单例子来熟悉一下拉普拉斯逆变换。首先考虑一个简单的指数函数：

```
>> syms s
>> ilaplace(1/s^3)
ans =
1/2*t^2
```

下面得到一个指数函数：

```
>> syms w
>> ilaplace(2/(w+s))
ans =
```

```
2*exp(-w*t)
```

最后这一个例子的结果含有三角函数：

```
>> ilaplace(s/(s^2+4))
ans =
cos(2*t)
```

当然，你在应用中遇到的很多问题并不一定像刚才这些例子一样有拉普拉斯逆变换结果，此时才是**MATLAB**真正有用的时候。举个例子，求下式的拉普拉斯逆变换：

$$F(s) = \frac{5 - 3s}{2 + 5s}$$

我们把它输入**MATLAB**：

```
>> F = (5-3*s)/(2+5*s);
```

计算它的拉普拉斯逆变换，我们求得**狄拉克δ函数**（对于工程师们是**单位脉冲**函数）：

```
>> ilaplace(F)
ans =
-3/5*dirac(t)+31/25*exp(-2/5*t)
```

这里是一个更复杂的例子。下式的拉普拉斯逆变换你认为会是什么？

$$F(s) = \frac{-s^2 - 9s + 4}{s^2 + s + 2}$$

现在，为了得到人能够阅读的形式你可以把自己埋在枯燥无味的代数运算中，或者你可以把它输进**MATLAB**中：

```
>> F = (-s^2 - 9*s + 4)/(s^2 + s + 2);
```

结果是：

```
>> ilaplace(F)
ans =
-dirac(t)-8*exp(-1/2*t)*cos(1/2*7^(1/2)*t)+20/7*7^(1/2)*exp(-1/2*t)*sin(1/2*7^(1/2)*t)
```

真是乱七八糟一大堆，但至少我们求得答案！

让我们尝试另一些很有可能在部分分式相乘时碰到的例子。例如：

```
>> F = 1/(s*(s+1)*(s+2));
>> ilaplace(F)
ans =
1/2+1/2*exp(-2*t)-exp(-t)
```

例9-1

$$F(s) = \frac{1}{(s + 7)^2}$$

的时间响应函数是什么？并绘制该函数的图象。

解9-1

拉普拉斯逆变换命令给出的结果是：

```
>> f = ilaplace(1/(s+7)^2)
f =
t*exp(-7*t)
```

注意，我们在这里是进行符号计算，因此我可以使用*ezplot*绘制它的图象：

```
>> ezplot(f)
```

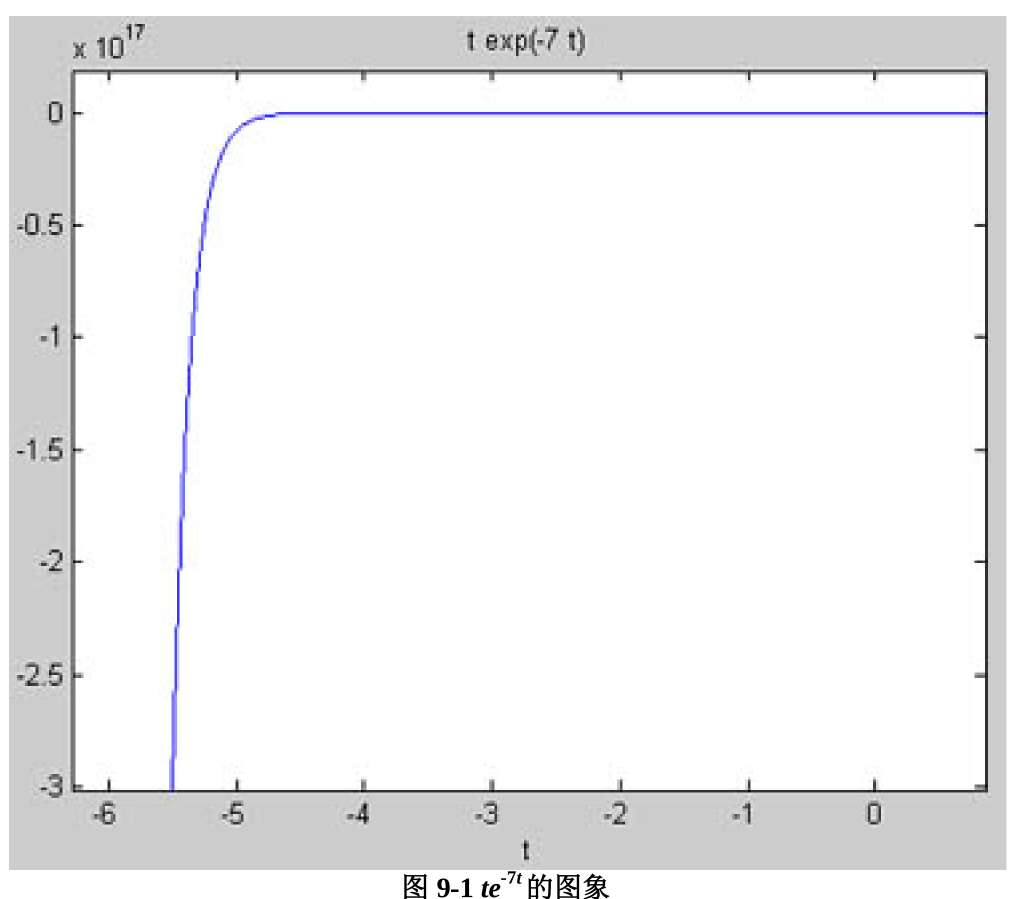


图 9-1 te^{-7t}的图象

结果显示在图9-1中。在应用中，我们可能仅对$t \geq 0$的情况感兴趣，因此我们让t为正值。首先我们了解函数在前5秒内的情况。输入如下：

```
>> ezplot(f,[0,5])
```

图象如图9-2所示。看起来所有的动作都在1秒前完成，然后函数迅速趋向零，因此这个图象还不符合要求。我们再试一次，仅看看$0 \leq t \leq 1$的情况：

```
>> ezplot(f,[0,1])
```

如图9-3,这一次我们终于把函数给漂亮地显示出来了。

例9-2

求下式的拉普拉斯逆变换并绘制它的图象。

$$\frac{2s + 3}{(s + 1)^2(s + 3)^2}$$

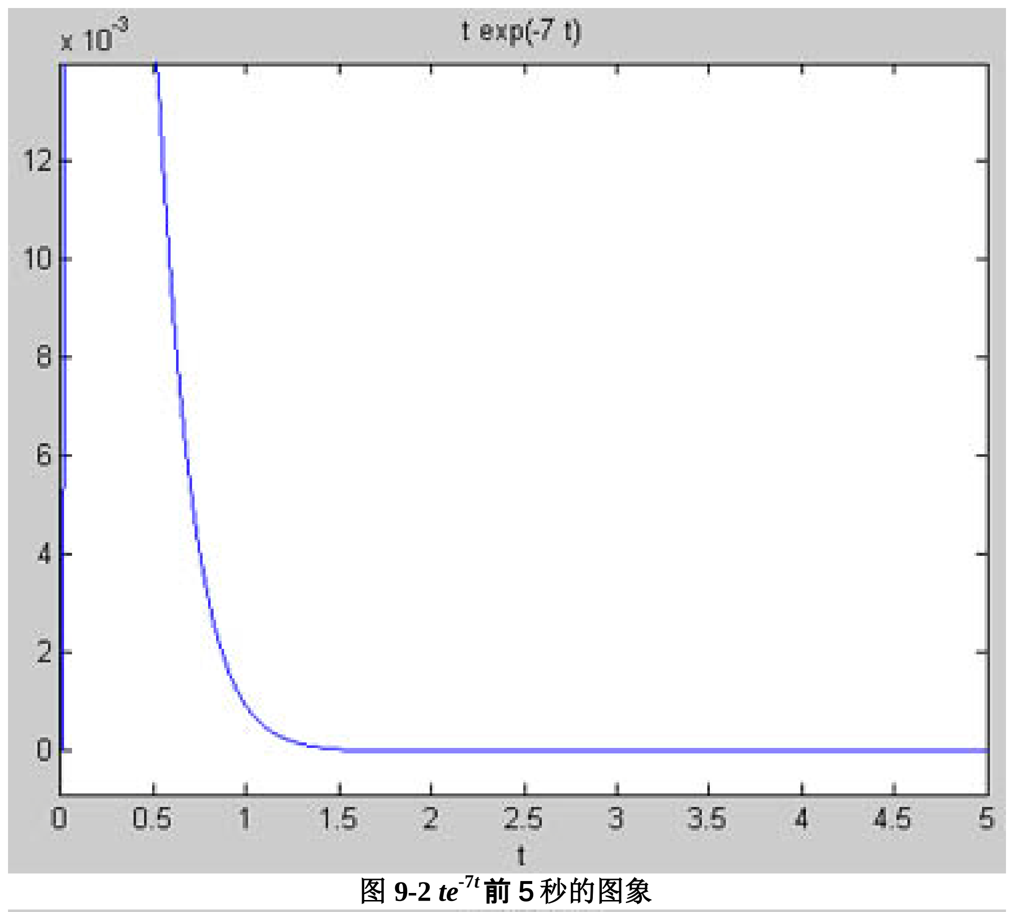


图 9-2 te^{-7t} 前 5 秒的图象

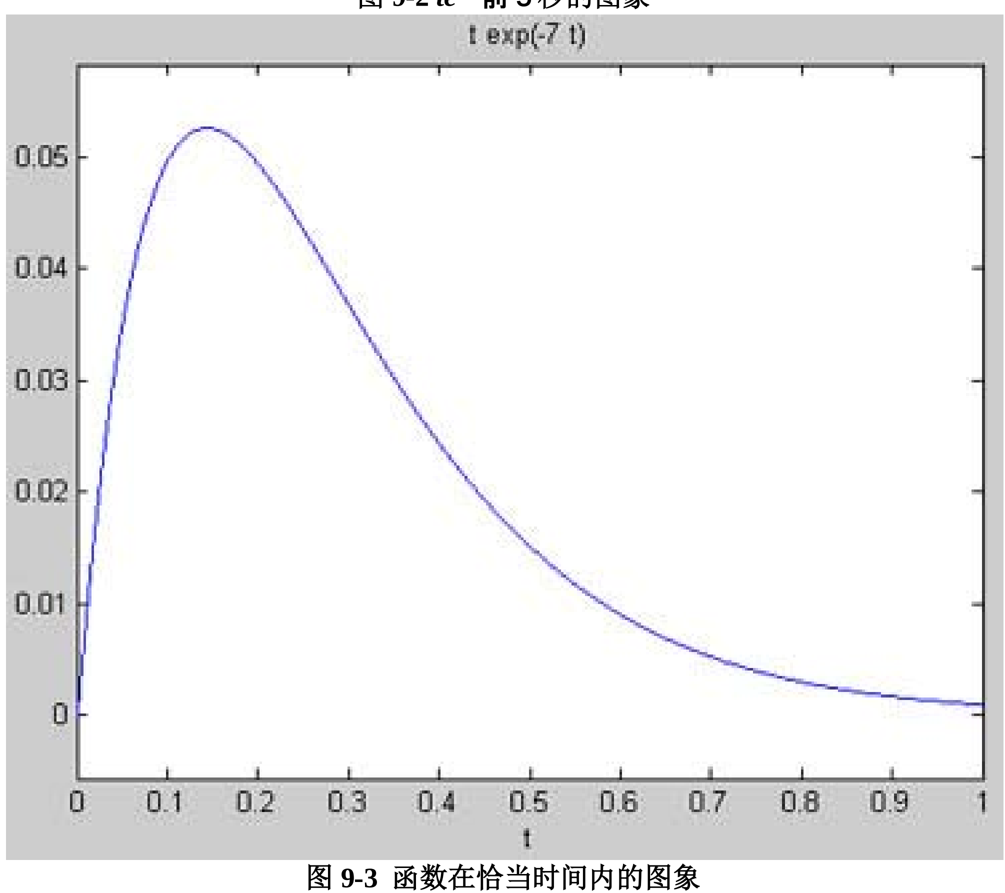


图 9-3 函数在恰当时间内的图象

解9-2

调用*ilaplace*我们求得：

```
>> syms s;
>> f = ilaplace((2*s+3)/((s+1)^2*(s+3)^2))
f =
exp(-2*t)*(-1/2*t*cosh(t)+(1/2+t)*sinh(t))
```

在本例中，我们发现在7秒内的图象很漂亮：

```
>> ezplot(f,[0,7])
```

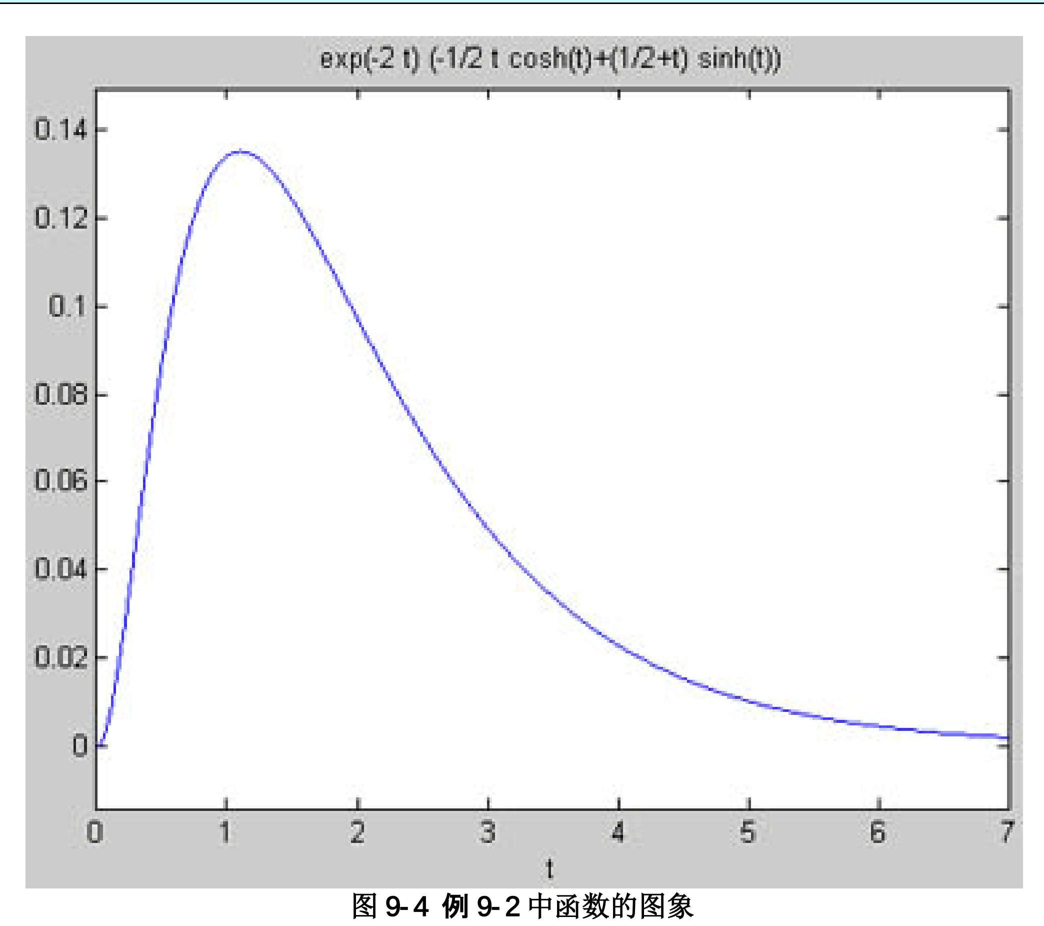


图 9-4 例 9-2 中函数的图象

重要的是，绘制图象可以把函数在t<0内的特性给显示出来，图象如图9-4所示。注意这个函数的图象形状与前一个例子一样——只不过响应的时间更长。

微分方程求解

拉普拉斯变换简化了微分方程，把它们变成代数方程。一个函数的一阶导函数的拉普拉斯变换是：

$$\mathscr{L}\left\{\frac{df}{dt}\right\} = sF(s) - f(0)$$

而函数的二阶导函数的拉普拉斯变换是：

$$\mathscr{L}\left\{\frac{d^2f}{dt^2}\right\} = s^2F(s) - sf(0) - f'(0)$$

不幸运的是，**MATLAB**并不按我想要方式工作。考虑我们在使用*dsolve*命令时在导数变量前加上*D*的事实。例如，我们可以求得一阶*ODE*的解：

```
>> dsolve('Dx = a*x')
ans =
C1*exp(a*t)
```

如果你可以使用拉普拉斯变换求得上面导数的结果是一件好事，不过它不能。例如，如果我们输入：

```
>> laplace(Dx - a*x)
```

我们会收到错误信息：

```
??? Undefined function or variable 'Dx'.
```

不要试着把它用字串形式输入：

```
>> laplace('Dx - a*x')
??? Function 'laplace' is not defined for values of class 'char'.
```

因此我们不能把*laplace*直接应用在导数上。要在**MATLAB**使用拉普拉斯变换方法求解微分方程，你不得不自己计算拉普拉斯变换，然后再输入到**MATLAB**中，这时才可以使用*ilaplace*求出解。我们用一个例子来演示。

例9-3

有两个质量弹簧系统，其中$m = 1$kg且满足下面的微分方程：

$$m\ddot{x}(t) + 1.2\dot{x}(t) + x = y(t) + \alpha\dot{y}(t)$$

求方程$X(s)$在定义域s的解并求它的逆变换$x(t)$。分别取常数α=0, 2, 5绘制它的图象。设$x(0) = \dot{x}(0) = y(0)$。假设$y(t)$由**单位阶跃**或**海维赛德**（*Heaviside*）函数给出。

解9-3

根据一阶导数和二阶导数的拉普拉斯变换法则，并设$m = 1$，我们得到下面s的方程：

$$s^2X(s) + 1.2sX(s) + X(s) = Y(s) + \alpha sY(s)$$

整理之后得到：

$$(s^2 + 1.2s + 1)X(s) = (1 + \alpha s)Y(s)$$

$$\Rightarrow X(s) = \frac{1 + \alpha s}{s^2 + 1.2s + 1}Y(s)$$

题意告诉我们y(t)是**单位阶跃**或**海维赛德**（*Heaviside*）函数。对于那些不熟悉这个函数的读者，它定义为：

$$y(t) = \begin{cases} 0 & t<0 \\ 1 & t>0 \end{cases}$$

在**MATLAB**中我们可以用下面的命令把它的图象绘制出来：

```
>> syms t
>> ezplot(heaviside(t),[-2,2])
```

结果显示在图9-5中。本质上是系统在t=0时刻有一个常量驱动的函数。

要解这道题，我们需要知道海维赛德函数的拉普拉斯变换：

```
>> syms t
>> laplace(heaviside(t))
ans =
1/s
```

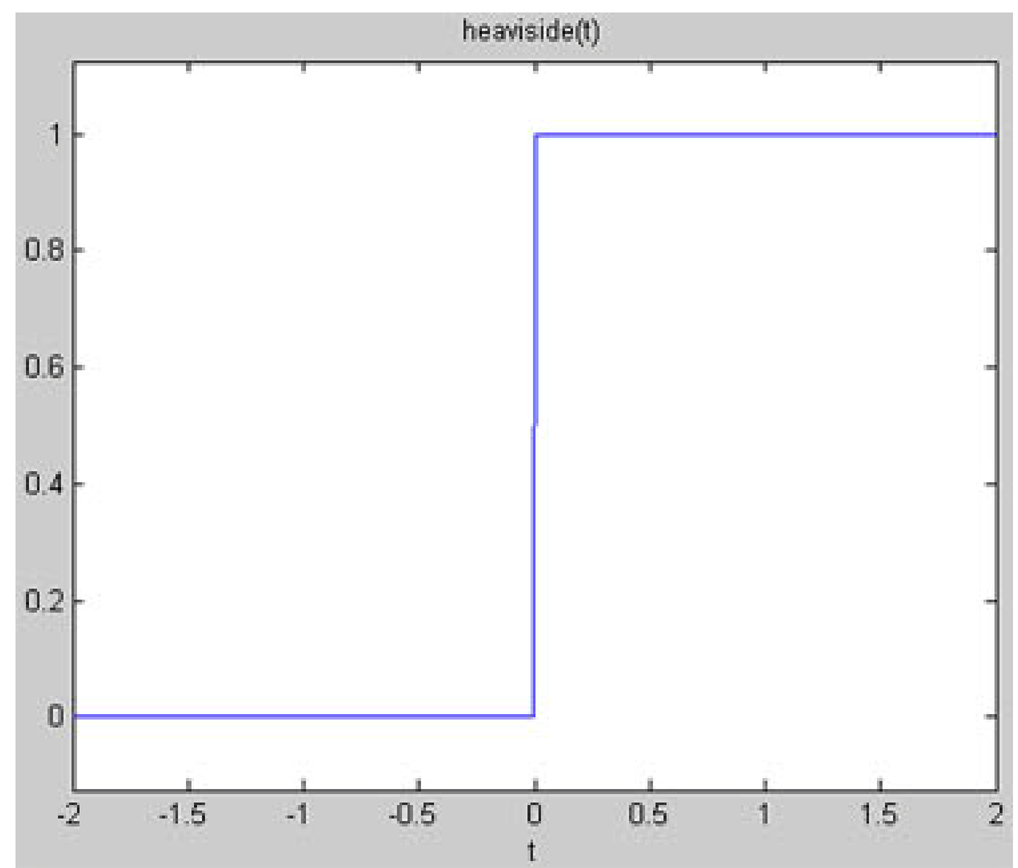


图 9-5 海维赛德或单位阶跃函数的图象

因此$x(t)$函数可以通过变换求得：

$$X(s) = \frac{1 + \alpha s}{s^2 + 1.2s + 1}\left(\frac{1}{s}\right)$$

我们把条件$\alpha = a, b, c$取$a = 0, b = 2, c = 5$：

```
>> a = 0; b = 2; c = 5;
```

现在我们把条件输进s。首先，为了简化输入，我们把三种情况的共同部分记为d：

```
>> syms s
>> d = s^2 + (1.2)*s + 1;
```

然后我们就能每个常量下的s函数定义为：

```
>> Xa = ((1+ a*s)/d)*(1/s);
>> Xb = ((1+ b*s)/d)*(1/s);
>> Xc = ((1+ c*s)/d)*(1/s);
```

现在我们计算逆变换：

```
>> xa = ilaplace(Xa)
xa =
-exp(-3/5*t)*cos(4/5*t)-3/4*exp(-3/5*t)*sin(4/5*t)+1
>>  xb = ilaplace(Xb)
xb =
-exp(-3/5*t)*cos(4/5*t)+7/4*exp(-3/5*t)*sin(4/5*t)+1
>>  xc = ilaplace(Xc)
xc =
```

```
1-exp(-3/5*t)*cos(4/5*t)+11/2*exp(-3/5*t)*sin(4/5*t)
```

$\alpha = 2$和$\alpha = 5$ 的情况很相似，因此我们只把$\alpha = 2$的图象与$\alpha = 0$并排绘出来。使用*subplot*命令即可。首先，我们告诉它我们需要1行2窗格，然后把第一个图象放到第一个窗格中：

```
>> subplot(1,2,1)
```

绘制第一个函数在$0 \le t \le 10$之间的图象：

```
>> ezplot(xa,[0 10])
```

现在我们告诉**MATLAB**要放置第二个图象：

```
>> subplot(1,2,2)
```

绘制第二个函数在$0 \le t \le 10$之间的图象：

```
>> ezplot(xb,[0 10])
```

结果如图9-6所示。

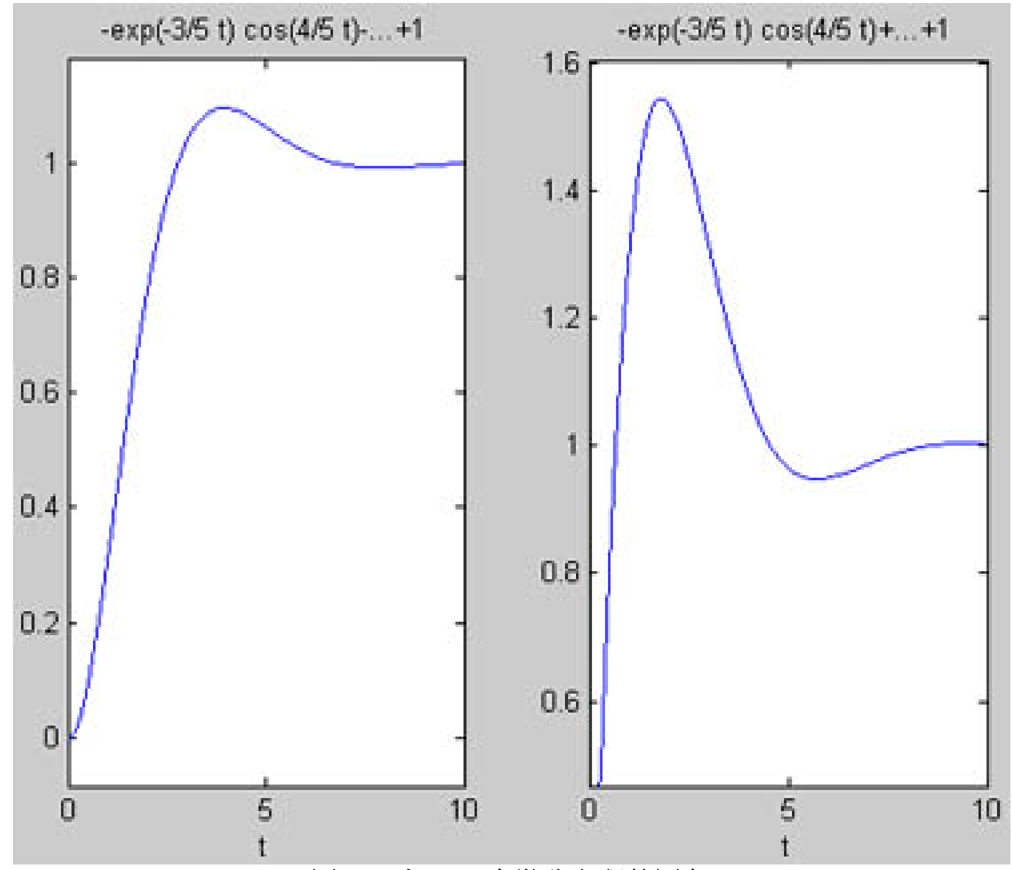


图 9-6 例 9-3 中微分方程的图象

傅立叶变换的计算

一个函数$f(t)$的傅立叶变换被定义为：

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt$$

在**MATLAB**中我们可以输入`fourier`命令计算一个函数的傅立叶变换。傅立叶变换允许你把时间或空间上的函数转换成频率上的函数。例如，我们可以使用傅立叶变换检验sin函数是由两个**狄拉克δ**构成：

```
>> syms x
>> fourier(sin(x))
ans =
i*pi*(-dirac(w-1)+dirac(w+1))
```

下面我们求高斯函数的傅立叶变换。首先我们定义函数并绘制它在“空间”定义域内的图象：

```
>> f = exp(-2*x^2);
>> ezplot(f,[-2,2])
```

图象如图9-7所示。

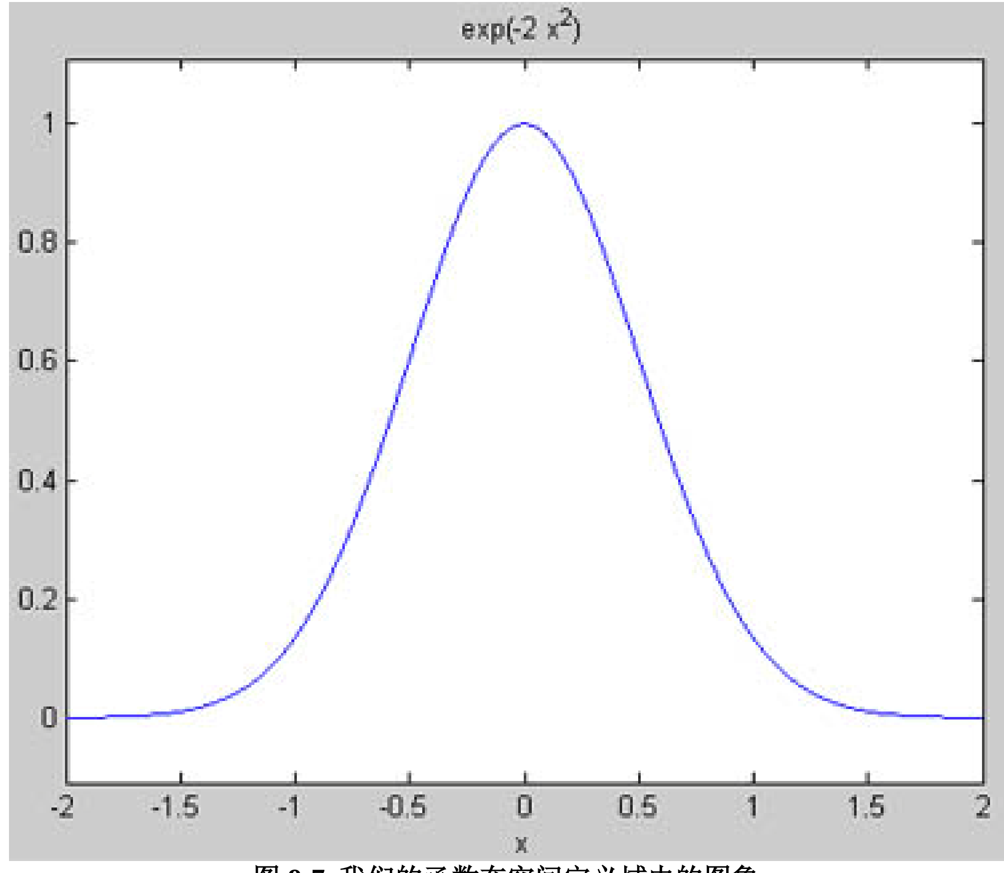


图 9-7 我们的函数在空间定义域内的图象

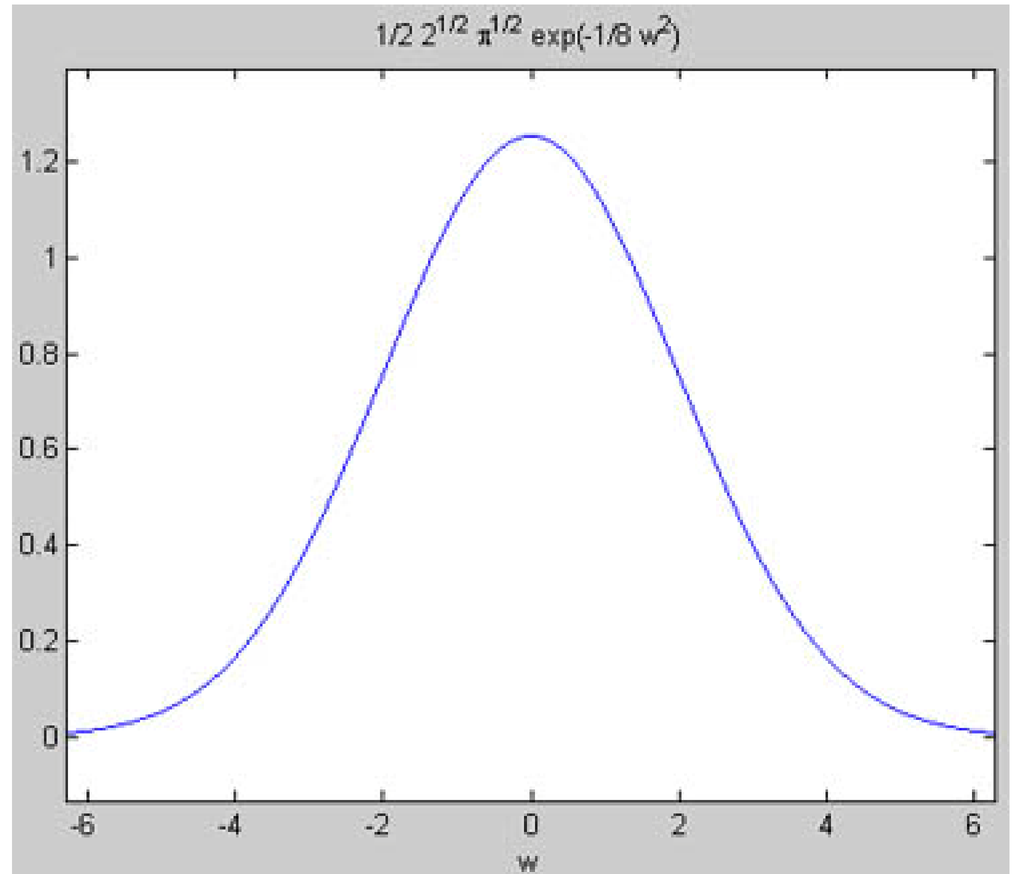


图 9-8 一个高斯的傅立叶变换是另一个具有不同高度和宽度的高斯

我们计算傅立叶变换：

```
>> FT = fourier(f)
FT =
1/2*2^(1/2)*pi^(1/2)*exp(-1/8*w^2)
```

我们发现一个高斯函数的傅立叶变换结果是另一个高斯函数——虽然带有缩放比例。通过比较图9-8与图9-7可从图象中看出，傅立叶变换（后的图象）的一些特性给显现出来了：函数的频率更宽、峰值更高。

下面是另一个好的例子。设有函数

$$f(x) = e^{-|x|} = \begin{cases} e^{x} & x<0 \\ e^{-x} & x>0 \end{cases}$$

在**MATLAB**中输入下面的命令定义它：

```
>> syms x;
>> f = exp(-abs(x));
```

绘制它的图象，结果如图9-9所示。现在我们计算它的傅立叶变换：

```
>> FT = fourier(f)
FT =
2/(1+w^2)
```

这个函数的图象如图9-10所示。

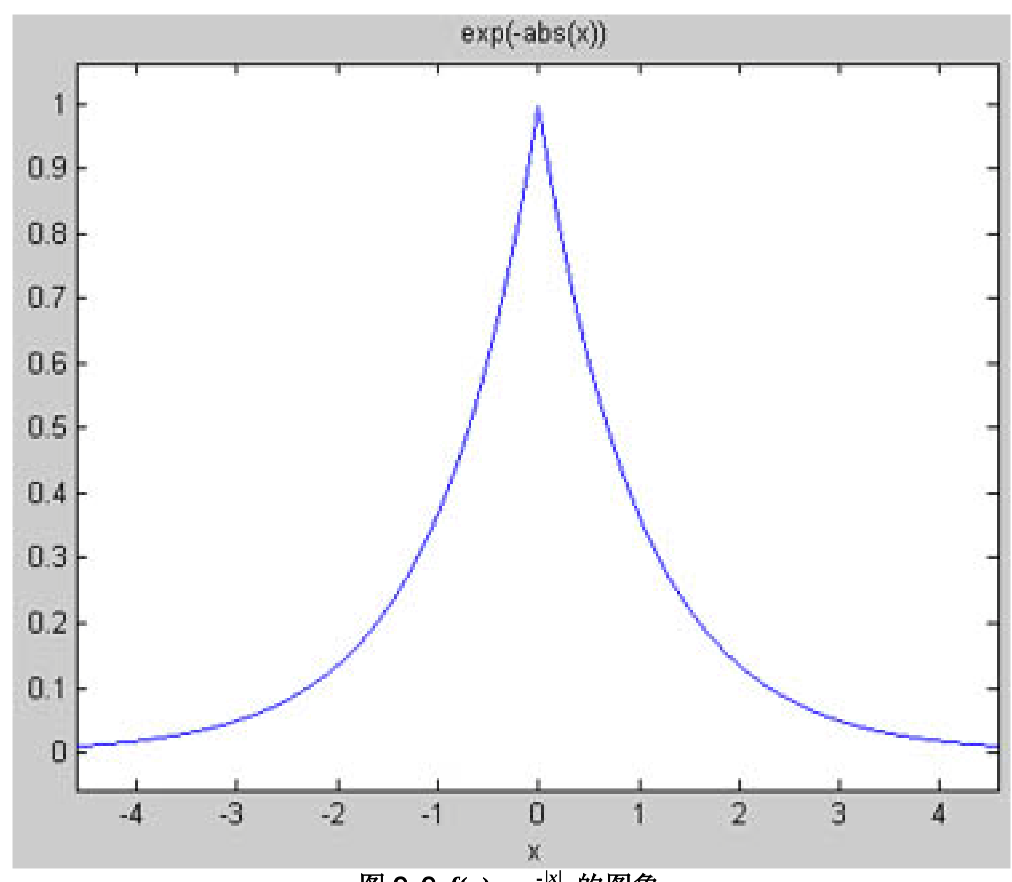


图 9-9 $f(x) = e^{-|x|}$ 的图象

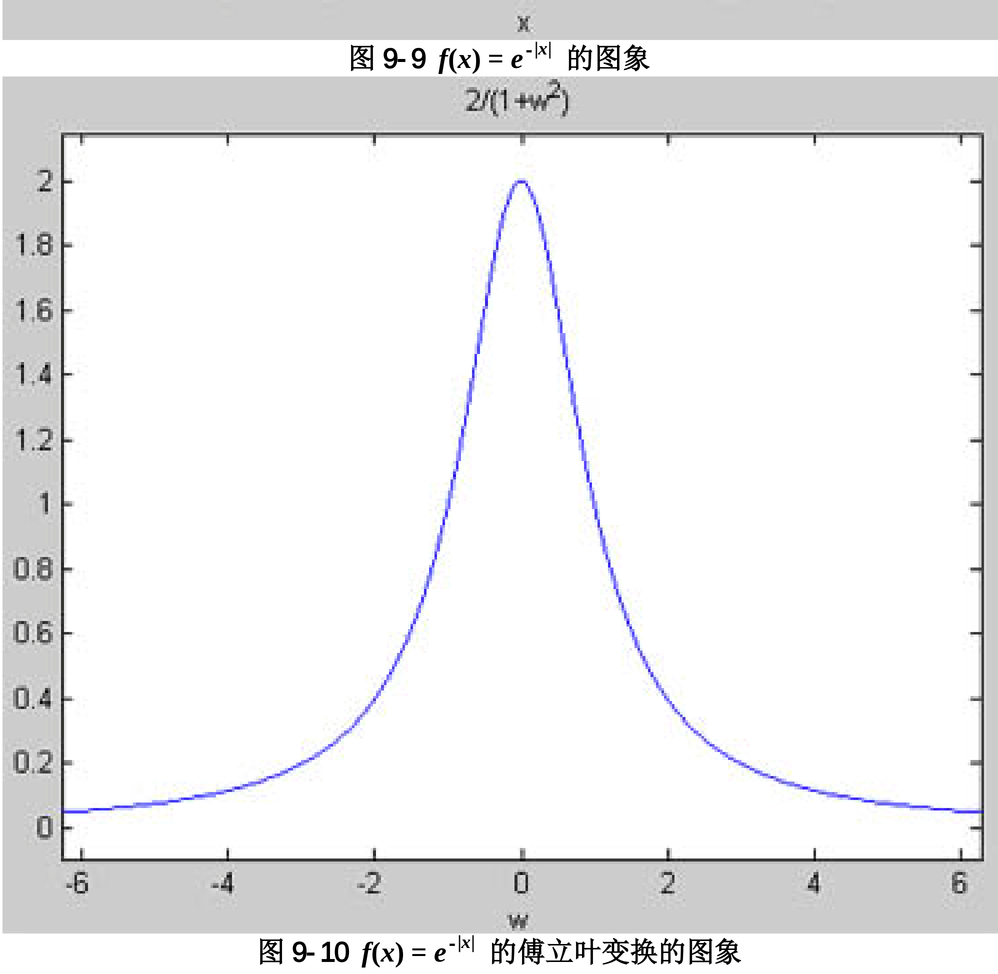


图 9-10 $f(x) = e^{-|x|}$ 的傅立叶变换的图象

傅立叶逆变换

要计算一个函数的傅立叶逆变换，使用的命令是*ifourier*。例如，我们可以输入下面的命令看出傅立叶变换的对偶性关系：

```
>> syms w
>> f = ifourier(-2*exp(-abs(w)))
```

结果是：

```
f =
-2/(1+x^2)/pi
```

这个函数显示在图9-11中。

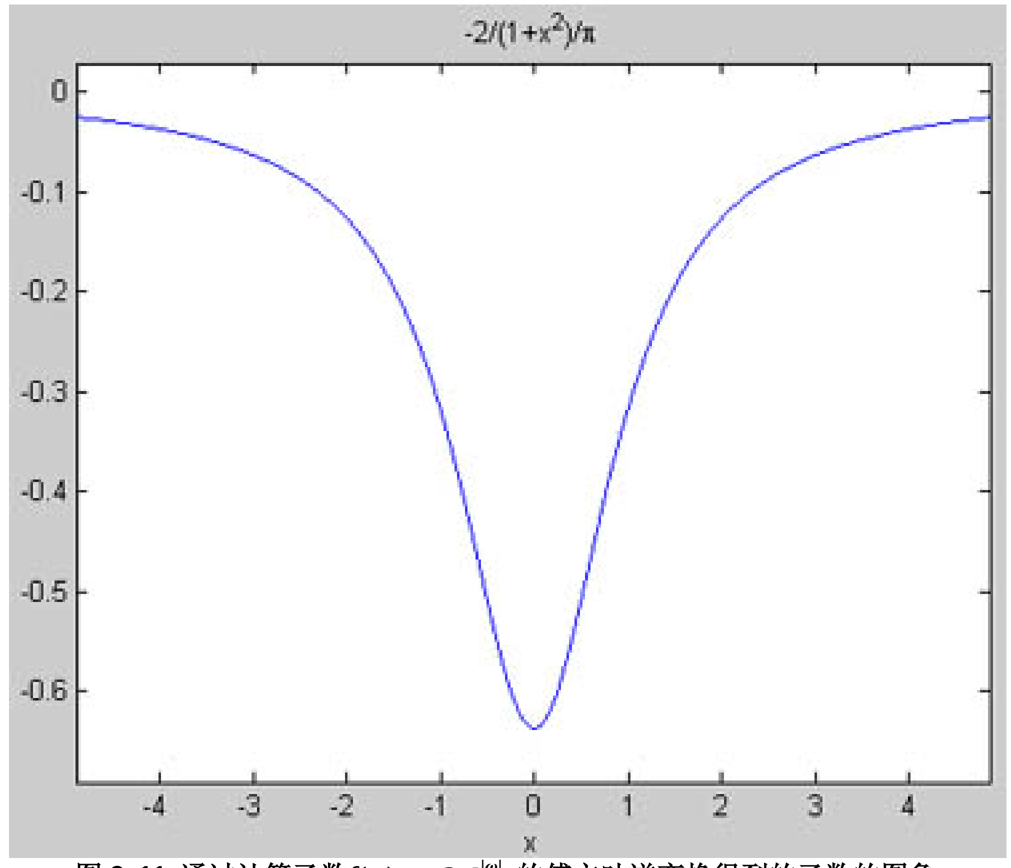


图 9-11 通过计算函数$f(\omega) = -2e^{-|\omega|}$ 的傅立叶逆变换得到的函数的图象

快速傅立叶变换

MATLAB的函数*fft*可以用来计算数值向量的**快速傅立叶变换**(Fast Fourier Transforms)。

例9-4

假设有一个信号$x(t) = 3\cos(\pi t) + 2\cos(3\pi t) + \cos(6\pi t)$。建立这个信号被噪声干扰的一个模型，然后计算*FFT*检查信号的频谱。时间选10秒。

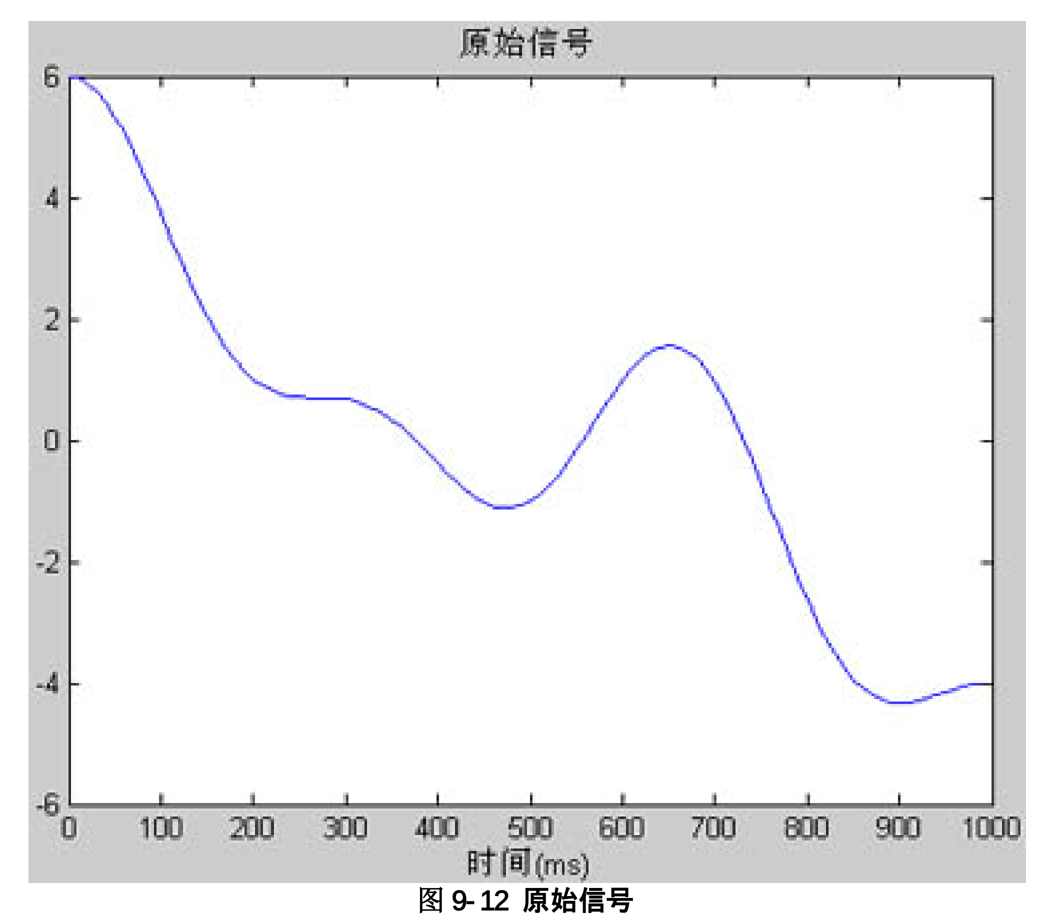


图 9-12 原始信号

解9-4

首先我们定义时间区间：

```
>> t = 0:0.01:10;
```

现在定义信号：

```
>> x = 3*cos(pi*t) + 2*cos(3*pi*t) + cos(6*pi*t);
```

为了模拟噪声，我们随机产生一些数字然后把它们添加到信号中去。要产生随机数字，我们可以调用*randn*来做：

```
>> x_noisy = x + randn(size(t));
```

我们绘制这两个图象，看看噪声的影响情况。在图9-12中，我们显示了前1000毫秒内未受干扰的原始信号图象。用来绘制这个图象的命令是：

```
>> plot(1000*t(1:100),x(1:100)), xlabel('时间(ms)'), title('原始信号')
```

现在我们绘制这一时间内的噪声信号。命令是：

```
>> plot(1000*t(1:100),x_noisy(1:100)),xlabel('时间(ms)'), title('噪声信号')
```

结果如图9-13所示。



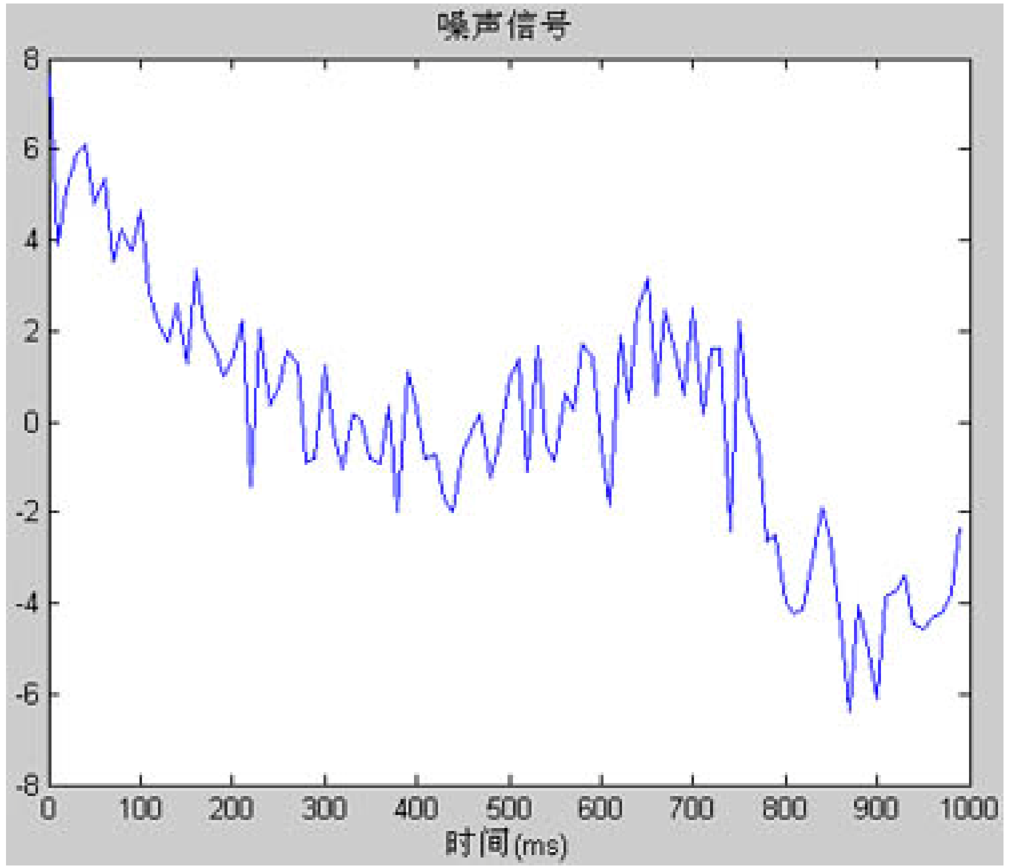


图 9-13 信号的噪声

噪声信号看起来真让人绝望。通过计算傅立叶变换，我们可以搜集原始信号的有用信息。输入*fft*(*f*, *n*)，我们能够计算函数*f*有*n*个点的快速傅立叶变换，仿效**MATLAB**帮助中给出的例子，我们计算函数有512个点的傅立叶变换：

```
>> FT = fft(x_noisy,512);
```

信号中的功率可以通过计算傅立叶变换和它的共扼复数的乘积再除以总点数求得：

```
>> P = FT.*conj(FT)/512;
```

当我们绘制函数时，信号的频谱就显现出来了。在图9-14中，我们看到三个尖峰，它们与$x(t) = 3\cos(\pi t) + 2\cos(3\pi t) + \cos(6\pi t)$中的三个频率相对应。注意原始信号中的幅度是如何在图9-14中的功率谱中反映出来的。

译者注：图9－14使用下面的命令绘制出来，虽然图形看起来一样，但其水平横坐标与原英文版（0：200）不同。因原书没有给出绘制该图的命令，不知是我绘制有问题还是原书有问题，请知道的读者告知。谢谢

```
>> f=t/0.01;
>> plot(f(1:50),P(1:50));
```

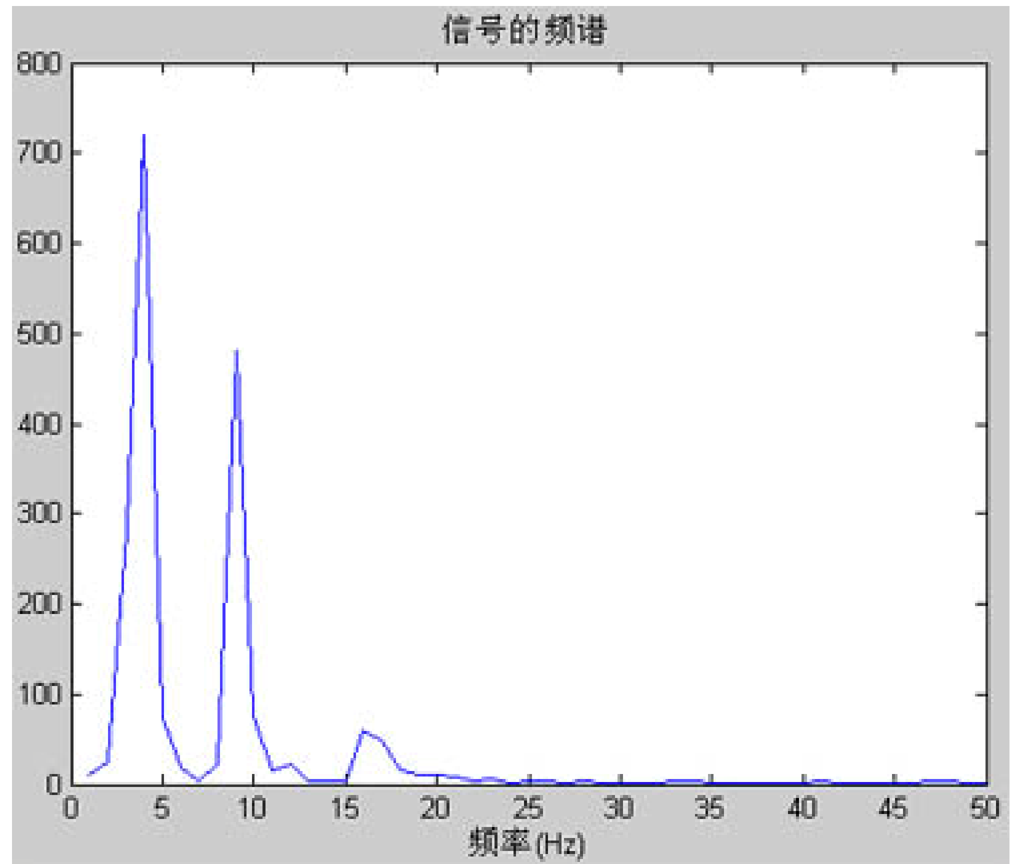


图9-14 通过计算噪声信号的离散傅立叶变换，我们能够提取信号里的频率信息。图中的三个尖峰及其相对强度是信号$x(t) = 3\cos(\pi t) + 2\cos(3\pi t) + \cos(6\pi t)$中的频谱的反映

习题

习题

1. 求$g(t) = te^{-3t}$的拉普拉斯变换。
2. 求$f(t) = 8\sin 5t - e^{-t}\cos 2t$的拉普拉斯变换。
3. $\frac{1}{s} - \frac{2s^2 + 3}{s^2 + 9}$ 的拉普拉斯逆变换是什么？
4. 求$\frac{s}{s^2 - 49} - \frac{3}{s^2 - 9}$ 的拉普拉斯逆变换。
5. 解微分方程：

$$\frac{dy}{dt} + y = 2U(t)$$

其中$U(t)$是海维赛德函数（单位阶跃函数）。设所有的初始条件都为零，求它的响应。
6. 求x^2的傅立叶变换。
7. 求$x\cos x$的傅立叶变换
8. 求$\frac{1}{1 + i\omega}$ 的逆变换。
9. 求$a = [2, 4, -1, 2]$的快速傅立叶变换。
10. 设$x(t) = \sin(\pi t) + 2\sin(4\pi t)$。使用$x + randn(size(t))$给信号添加噪声，则在哪个频率上信号的功率谱最大？

【参考答案在第 240 页】



第十章

曲线拟合

MATLAB 可以用来求与一组数据最吻合的函数。为了达到目的，本章我们将学习一些简单的技术。

线性函数拟合

我们想到的最简单的情况是由线性函数很好描述的一个数集，也就是说如果我们所考虑的数据是以 $y = f(x)$ 的形式给出，并且其中 $f(x)$ 满足：

$$y = mx + b$$

要求得 m 和 b 的值，我们可以使用一个称为 *polyfit*(*x*, *y*, *n*)的 **MATLAB** 函数，其中 n 是我们要 **MATLAB** 求出的多项式的次数，对于 $y = mx + b$ 形式的方程，我们把 n 设为等于 1，因此调用的语句将是 *polyfit*(*x*, *y*, 1)。*polyfit* 函数采用最小二乘法计算。让我们使用一些简单的例子看看如何使用它。

例 10-1

在一组打高尔夫球选手中，差点和平均成绩存在如下的关系：

差点	6	8	10	12	14	16	18	20	22	24
平均成绩	3.94	3.8	4.1	3.87	4.45	4.33	4.12	4.43	4.6	4.5

求拟合这些数据的一条曲线并评估它的拟合程度。

解 10-1

解这个问题并不需要高尔夫的相关知识，你所需要知道的是我们设想差点与平均成绩之间存在线性关系，而我们要求得一个方程来描述它。一旦你有了方程 $y = mx + b$，你就可以预知你没有的 x 所对应的 y 值。首先我们把这两组数据输进 **MATLAB**：

```
>> handicap = [6:2:24]
handicap =
     6     8    10    12    14    16    18    20    22    24
>> Ave = [3.94, 3.8, 4.1, 3.87, 4.45, 4.33, 4.12, 4.43, 4.6, 4.5];
```

接着我们调用 *polyfit* 让 **MATLAB** 计算拟合数据的多项式的系数。要让 **MATLAB** 产生 $y = mx + b$ 形式的一次多项式，首先我们需要确定 x（独立变量）和 y（因变量）分别是什么。在本例中充当 x 角色的独立变量是差点(*Handicap*)，而充当 y 角色的独立变量是平均成绩(*Average*)。既然我们想要产生一个一次多项式，我们用下面的方式调用 *polyfit*：

```
>> p = polyfit(handicap,Ave,1);
```

下一步我们需要提取出 **MATLAB** 求得的系数，一般来说，**MATLAB** 会以下面的方式产生多项式的系数：

$$p(x) = p_1x^n + p_2x^{n-1} + \ldots + p_{n-1}x^2 + p_nx + p_{n+1}$$

如果我们使用 *p* = *polyfit*(*x*, *y*, *n*)调用 *polyfit*，*p*(*j*)即引用第 j 个系数。在我们的例子中，*polyfit* 是以 *p*(1)**x* + *p*(2)的形式返回方程的。因此我们提取系数的方法是：

```
>> m = p(1)
m =
    0.0392
>> b = p(2)
b =
    3.6267
```

现在我们产生一个函数绘出 $y = mx + b$ 的直线。第一步我们需要建立 x 轴：

```
>> x = [6:0.1:24];
```

然后创建绘制直线的函数：

```
>> y = m*x + b;
```

我们也沿着直线把实际数据标出来，它们将各自以独立的点显示。使用下面的命令，我们把这些数据用小圆圈表示：

```
>> subplot(2,1,2);
>> plot(handicap,Ave,'o',x,y),xlabel('差点'),ylabel('平均成绩')
```

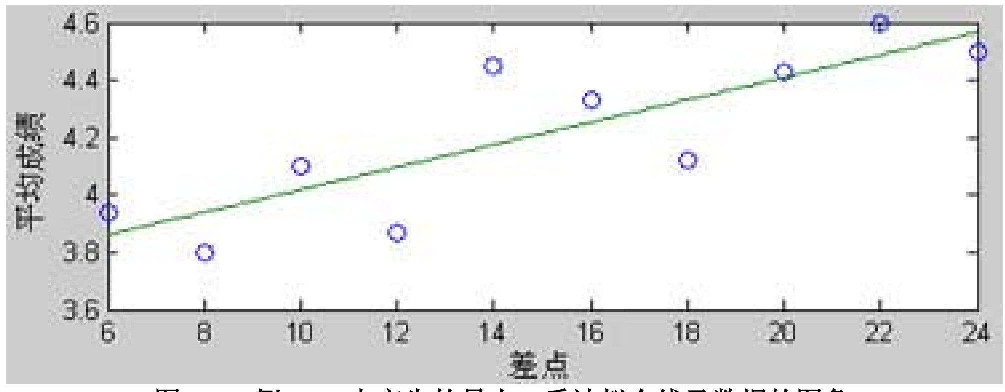


图 10-1 例 10-1 中产生的最小二乘法拟合线及数据的图象

结果如图 10-1 所示。不过拟合并不完美，很多数据点分布在我们产生的直线较远的地方。让我们看看本例中由拟合直线对特定的差点所预计的成绩值的情况，这可以通过执行下面的命令计算：

```
>> w = m*handicap + b
w =
  Columns 1 through 6
    3.8616    3.9399    4.0182    4.0965    4.1748    4.2532
  Columns 7 through 10
    4.3315    4.4098    4.4881    4.5664
```

回头看刚才的原始数据表格，我们看到这些数据还是相当接近的。由于这些数据是使用平滑直线 $y = mx + b$ 产生的，因此 w 中保存的数据通常称为**平滑数据**。

在很多情况下测量得到的数据都带有误差，而方程 $y = mx + b$ 可以比原始数据更好地表达出变量之间的实际关系。

现在让我们了解一下拟合达到了什么程度。我们可以通过计算误差来评估拟合程度。第一项可以用来了解拟合程度的是残差。假设我们要把函数 $f(x)$ 与数集 t_i 进行拟合，残差平方的总和计算公式是：

$$A = \sum_{i=1}^{N}[f(x_i) - t_i]^2$$

现在用 $\bar{t}$ 表示数集的均值。数集与均值偏差的平方和计算公式是：

$$S = \sum_{i=1}^{N}(x_i - \bar{t})^2$$

那么 r 平方值就是：

$$r^2 = 1 - \frac{A}{S}$$

如果 $r^2 = 1$，那么函数将与数据完美拟合，因此 r^2 越接近 1 拟合就越好。在 **MATLAB**，我们可以相当简单地实现这些公式。首先，我们计算这些数据的均值，一共有 10 个数据点，

因此：

```
>> N = 10;
>> MEAN = sum(Ave)/N
MEAN =
    4.2140
```

我们可以使用 **MATLAB** 的内置函数计算数组中的数据的均值：

```
>> mean(Ave)
ans =
    4.2140
```

然后：

```
>> S = sum((Ave - MEAN).^2)
S =
    0.7332
```

(不用对 *Ave* 的使用感到不解，*Ave* 是一个包含平均成绩的数组)。现在我们计算 *A*，此时需要相关数据点的拟合值——我们前面创建的数组 *w* 就是了，因此：

```
>> A = sum((w - Ave).^2)
A =
    0.2274
```

现在我们计算 r^2：

```
>> r2 = 1 - A/S
r2 =
    0.6899
```

结果 *r* 平方值并不怎么接近 1，因此并不是很完美的拟合，不过也不太差，因为与 0 相比它更接近 1。

让我们尝试另一个更加有难度的例子。

例 10-2

下面的表格列出了缅因州欢乐谷房子的平方英尺数与其对应的平均售价。求拟合这些数据的一个线性函数。

平方英尺数	1200	1500	1750	2000	2250	2500	2750	3000	3500	4000
平均售价(千)	$135	$142	$156	$165	$170	$220	$225	$275	$300	$450

解 10-2

首先我们把数据输入两个数组：

```
>> sqft = [1200,1500,1750,2000,2250,2500,2750,3000,3500,4000];
>> price = [135,142,156,165,170,220,225,275,300,450];
```

现在我们绘制数据的图象：

```
>> plot(sqft,price,'o'),xlabel('房子平方英尺数'), ylabel('平均售价'),...
title('欢乐谷房子平方英尺数的平均售价'), axis([1200 4000 135 450])
```

图象如图 10-2 所示。

虽然 4000 平方英尺的房子的价格看起来有点偏离正常，其它的数据还是基本上在一条



直线的周围的，让我们找出最拟合这些数据的一条直线。在我们尝试求出 $y = mx + b$ 的过程中，房子的 *SQFT*（平方英尺数）充当 x 的角色而平均售价充当 y 的角色。使用 *polyfit* 找出我们需要的系数，只需把数据传递给它并告知它我们在求一次的多项式。调用的语句是：

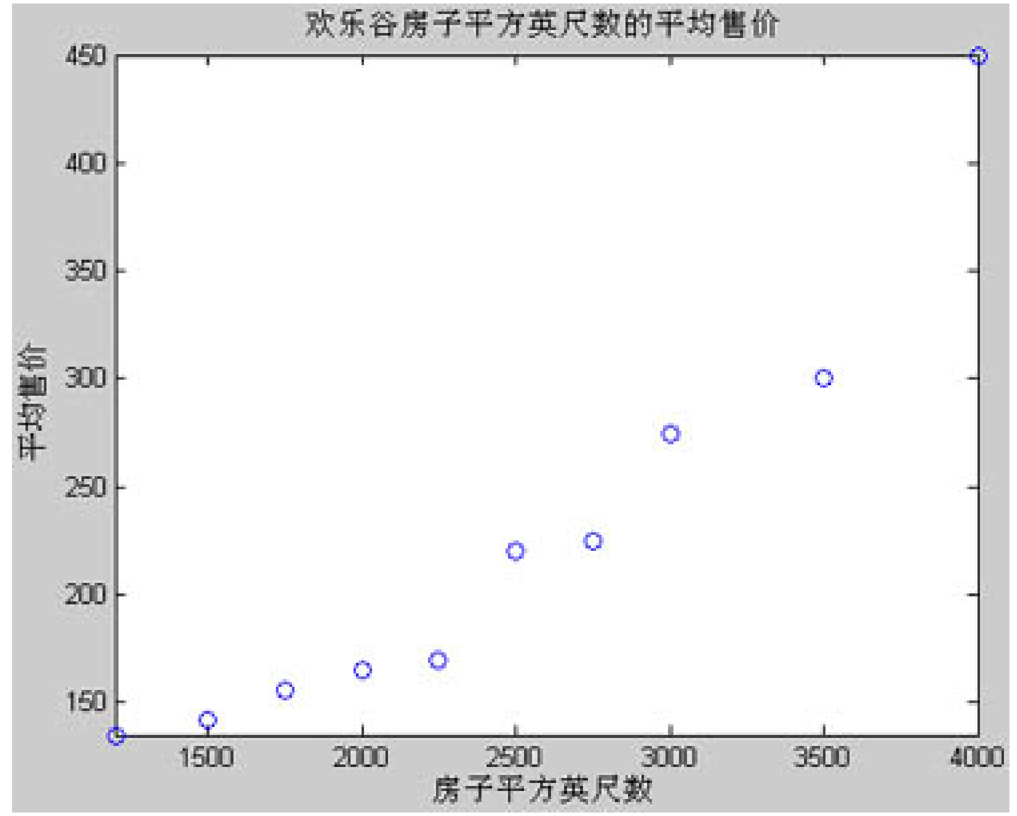


图 10-2 例 10-2 中使用的数据

```
>> p = polyfit(sqft,price,1);
```

在本例中 *polyfit* 函数返回两个元素。它们是 p(1) = m 和 p(2) = b。我们取出这些值：

```
>> m = p(1)
m =
    0.1032
>> b = p(2)
b =
  -28.4909
```

下一步我们就可以绘制 **MATLAB** 得到的拟合直线，并把它和实际数据点比较。首先我们产生 x 轴所用的数据：

```
>> x = [1200:10:4000];
```

现在我们使用上面求得的系数产生 y：

```
>> y = m*x + b;
```

最后，我们调用 *subplot*，把两条曲线在同一个图上显示出来：

```
>> subplot(2,1,2);
>> plot(x,y,sqft,price,'o'),xlabel('房子平方英尺数'), ylabel('平均售价'), ...
```

```
title('欢乐谷房子平方英尺数与平均售价'), axis([1200 4000 135 450])
```

与直线拟合的数据同时也在图 10-3 中显示出来。很多价格数据在很大程序上与直线相靠近，但最小和最大建筑面积不是，即实际上这并不是那么很好的拟合。*polyfit* 程序确定了能最好描述这些数据的函数是：

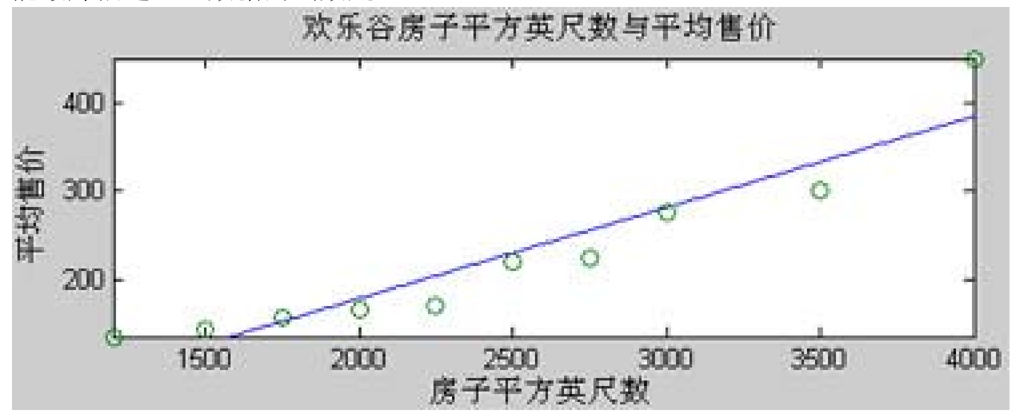


图 10-3 MATLAB 产生的拟合例 10-2 中的数据的图象

$$y = (0.1032)x - 28.4909$$

其中 x 是房子的建筑面积而 y 是以千为单位的房子价格。这个模型将预计一个 1600 平方英尺的房子的售价是：

$$y = 0.1032\times1600 - 28.4909 = 136.63$$

或者说大约$136,000。参考表格中的数据，可以看出这还是有点差距的。另一种我们可以用来体现拟合程度的方法是计算一个称为**均方根（Root-Mean-Square，RMS）误差**的数值，要用过这种方法来评估多项式与实际数据的拟合程度，我们可以使用下面的命令：

```
>> w = m*sqft + b
```

产生的数据和原始数据一起写在这里：

平方英尺数	1200	1500	1750	2000	2250	2500	2750	3000	3500	4000
平均售价(千)	$135	$142	$156	$165	$170	$220	$225	$275	$300	$450
预计值	$95	$126	$152	$178	$204	$230	$255	$281	$333	$384

你可以从一些例子中看出，误差并不那么明显。例如，对于个 2500 平方英尺数的房子来说，预计值与实际售价的差的百分比是：

$$\frac{|\$230 - \$220|}{\$230} = 0.0455 \approx 4\%$$

而对于一个 1500 平方英尺的房子：

$$\frac{|\$142 - \$126|}{\$142} = 0.1127 \approx 11\%$$

房地产经纪人可能并不要求很高的精度，因此 4%的误差是可以接受的，但 11%的误差可能就没法让人接受了。现在让我们计算 *RMS* 误差，以便对这个模型有更好的理解。要计算 *RMS* 误差，我们应用下面的公式：

$$RMS\ error = \left[\sum_{i=0}^{N}\frac{(T_i - A_i)^2}{N}\right]^{1/2}$$

其中 T_i 是准确值，A_i 是模型的近似值或预计值，而 N 是数据点的总个数。首先我们计算差额：

```
>> d = price - w;
```

模型的数据点总个数为 $N = 10$：

```
>> N = 10;
```

为了求出 *RMS* 误差，需要把差额进行平方。我们使用下面的命令把差额数组中的每个元素进行平方：

```
>> d2 = d.^2;
```

现在我们可以求 *RMS* 误差了：

```
>> RMS = sqrt((1/N)*sum(d2))
RMS =
   30.9322
```

确实是一个非常大的误差！*RMS* 误差本来应该是小于 1 的。如何改善这种状况呢？我们可以尝试拟合更高阶的多项式。让我们使用一个二次多项式看看。使用下面的步骤来做：

```
>> p = polyfit(sqft,price,2);
```

这次有三个系数产生。次数为 2 的 *polyfit* 函数使用下面的形式给我们返回系数：

$$y = p_1x^2 + p_2x + p_3$$

我们把它们提取出来放进变量中并绘图：

```
>> a = p(1);
>> b = p(2);
>> c = p(3);
>> x = [1200:10:4000];
>> y = a*x.^2+ b*x + c;
>> plot(x,y,sqft,price,'o'), xlabel('房子平方英尺数'),ylabel('平均售价'), ...
title('欢乐谷的房子平均售价与平方英尺数的关系'), axis([1200 4000 135 450])
```

图象如图 10-4 所示。从中可以看出，这个模型能更好地拟合数据，虽然它还未能达到完美。

现在我们求 *RMS*：

```
>> d1 = (w - price).^2;
>> w = a*(sqft.^2) + b*sqft + c
w =
  Columns 1 through 6
  139.5689  142.8071  150.6661  163.2164  180.4581  202.3912
  Columns 7 through 10
  229.0156  260.3314  337.0371  432.5081
>> RMS2 = sqrt((1/N)*sum(d1))
RMS2 =
   15.4324
```

这次 *RMS* 减少了一半，有了大改善。注意检查 *w* 数组中的元素，它的预计值与真实值更加接近了。让我们计算 *r* 平方值：

```
>> M1 = mean(price)
M1 =
  223.8000
>> S = sum((price - M1).^2)
S =
```

```
    8.5136e+004
>> w = a*sqft.^2 + b*sqft + c;
>> A = sum((w-price).^2)
A =
    2.3816e+003
>> r2=1-A/S
r2 =
      0.9720
```

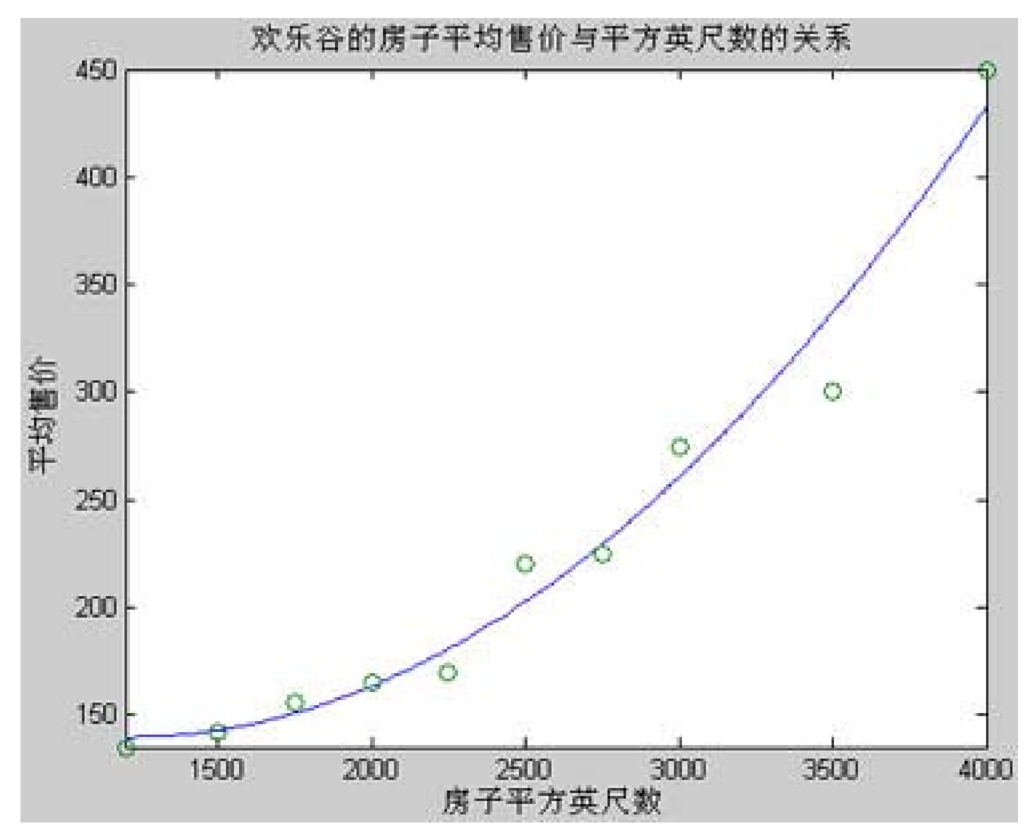


图 10-4 使用二次多项式之后提高了拟合程度

在二次多项式拟合这种情况下 *r* 平方的数值已经相当接近 1 了。所以我们可以认为这种模型是非常恰当的。

例 10-3

一金属块加热到 300°F，在接下来 7 小时内的温度如下：

时间(h)	温度(F)
0	300
0.5	281
1.0	261
1.5	244
2.0	228
2.5	214
3.0	202
3.5	191
4.0	181
5.0	164
6.0	151
6.1	149
7.0	141

试使用三次多项式拟合数据并评估拟合程度：

解 10-3

我们在 **MATLAB** 输入数据：

```
>> time = [0,0.5,1.0,1.5,2,2.5,3,3.5,4,5,6,6.1,7];
>> temp = [300,281,261,244,228,214,202,191,181,164,151,149,141];
```

让我们绘制数据的图象，如图 10-5 所示。

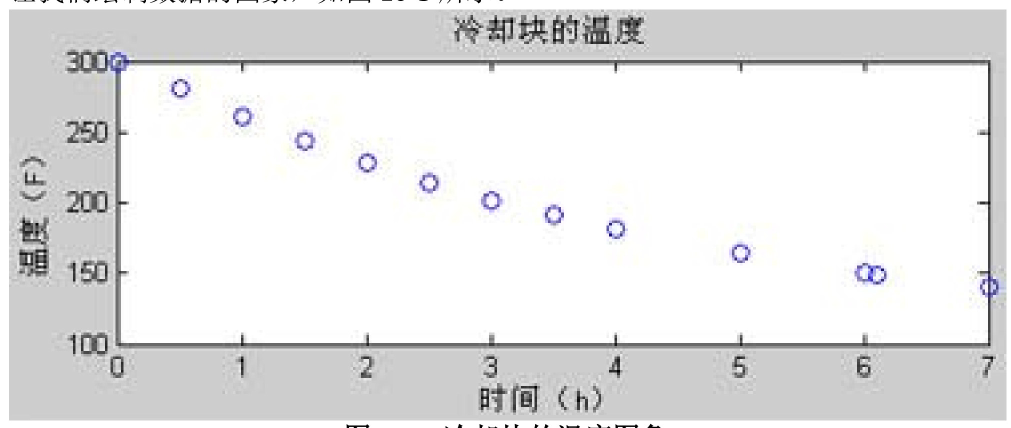


图 10-5 冷却块的温度图象

```
>> subplot(2,1,1), plot(time,temp,'o'),xlabel('时间（h）'), ...
ylabel('温度（F）'),title('冷却块的温度');
```

现在我们调用 *polyfit* 产生三次多项式的系数：

```
>> p = polyfit(time,temp,3);
```

既然多项式是三次的，我们的拟合函数具有下面的形式：

$$y = p_1x^3 + p_2x^2 + p_3x + p_4$$

现在我们提取系数：

```
>> a = p(1); b = p(2); c = p(3); d = p(4);
```

让我们为时间轴创建数据：

```
>> t = linspace(0,7);
```

接着定义拟合函数：

```
>> y = a*t.^3 + b*t.^2 + c*t + d;
```

我们还需要一组确切时间点上的温度数据：

```
>> w = a*time.^3 + b*time.^2 + c*time + d;
```

让我们在同一个图中绘制数据和拟合曲线的图象：

```
>> plot(time,temp,'o',t,y),xlabel('时间(h)'), ...
ylabel('温度(F)'),title('金属块冷却的三阶拟合图象')
```

结果如图 10-6 所示。看起来这个三次多项式能够很好地拟合。

让我们计算 r^2。首先我们计算数据的均值和 S：

```
>> M1 = mean(temp)
M1 =
  208.2308
>> S = sum((temp - M1).^2)
S =
  3.2542e+004
```

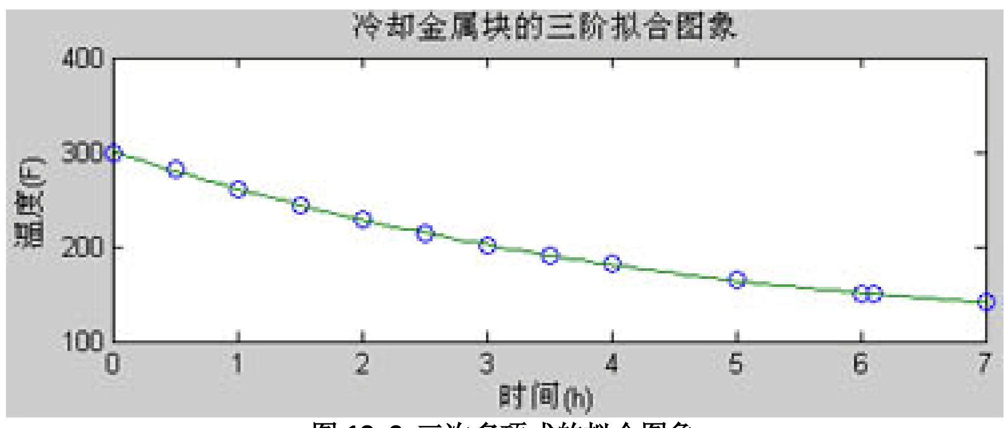


图 10-6 三次多项式的拟合图象

现在我们求 *A*：

```
>> A = sum((w-temp).^2)
A =
    2.7933
```

我们求得的 r^2 是：

```
>> r2 = 1 - A/S
r2 =
    0.9999
```

看起来与数据完美拟合。然而这是一个误导，事实上描述这些数据的真实函数是：

$$T = 100 + 202.59e^{-0.23t} - 2.59e^{-18.0t}$$

让 **MATLAB** 以 *long* 格式输出 *r* 平方的值：

```
>> format long
>> r2
r2 =
   0.99991416476050
```

是的，正如你从图中所看到的，拟合非常精确。让我们计算 *RMS*。

```
>> N = 13;
>> RMS = sqrt(sum((1/N)*(w-temp).^2))
RMS =
   0.46353796413735
```

RMS 值比 1 还小，因此我们可以认为它是一个很好的近似。既然我们有函数 $y(t)$，我们就能够估算出还没取得数据的任意时刻的温度。例如，在上面 7 小时之外的温度值。让我们计算经过更长时间的函数值。首先我们延伸时间线：

```
>> t = [0:0.1:15];
```

重新产生拟合多项式：

```
>> y = a*t.^3 + b*t.^2 + c*t + d;
```

我们可以使用 *find* 命令提问与数据有关的问题。例如，温度小于 80 度是在什么时候，因为我们有可能会冒险拿起这个金属块：

```
>> find(y < 80)
ans =
   148   149   150   151
```

find 命令返回了满足这个要求的数组。我们可以使用这些位置引用时间数组 *t* 和温度数组 *y* 中的数据。首先我们提取这些时刻，并在行末添加单引号把它们转置成列向量：

```
>> format short
>> A = t(148:151)'
A =
   14.7000
   14.8000
   14.9000
   15.0000
```

现在我们取相对应的温度：

```
>> B = y(148:151)'
B =
   78.5228
   77.0074
   75.4535
   73.8604
```

我们可以把数据放进一个两列的表格中，左边一列是时间，右边一列是温度：

```
>> Table = [A B]
Table =
   14.7000   78.5228
   14.8000   77.0074
   14.9000   75.4535
   15.0000   73.8604
```

举个例子，我们可以看到在 14.9 小时这个物块的预计值约为 75 度。现在让我们产生从 10 小时到 15 小时之间的数据的图象。首先我们求 10 小时这个时刻在数组中的索引。我们已经以 0.1 步长产生时间数据，因此我们可以搜索 *t*>9.9 的情况：

```
>> find(t > 9.9)
ans =
  Columns 1 through 11
   101   102   103   104   105   106   107   108   109   110   111
  Columns 12 through 22
   112   113   114   115   116   117   118   119   120   121   122
  Columns 23 through 33
   123   124   125   126   127   128   129   130   131   132   133
  Columns 34 through 44
   134   135   136   137   138   139   140   141   142   143   144
  Columns 45 through 51
```

```
    145   146   147   148   149   150   151
```

接着我们创建一个新的数组来包含这些时间值：

```
>> T1 = t(101:151);
```

现在让我们攫取这个范围内的温度值并储存它们：

```
>> TEMP2 = y(101:151);
```

这时我们就可以绘制它们的图象了：

```
>> plot(T1,TEMP2),xlabel('时间(h)'),ylabel('温度(F)'), axis([10 15 60 120])
```

结果如图 10-7 所示。这个模型预言在 5 小时内金属块的温度将下降 40 多度。

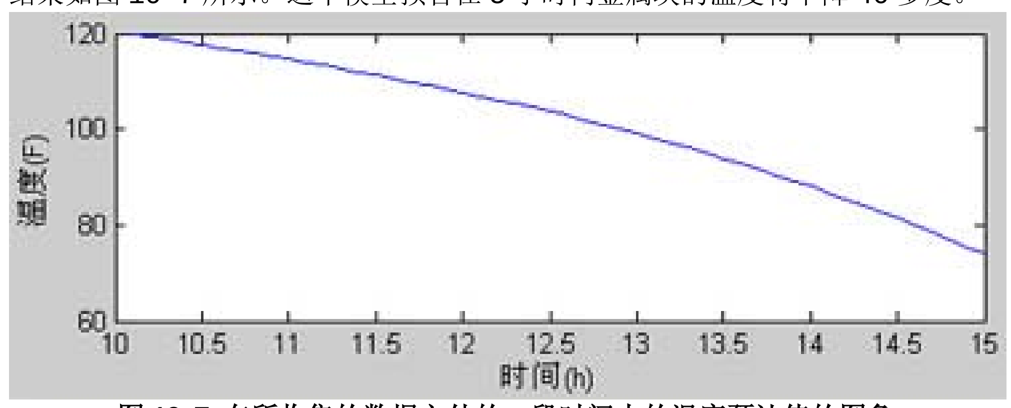


图 10-7 在所收集的数据之外的一段时间内的温度预计值的图象

指数函数的拟合

除了多项式之外还存在着其它类型函数拟合的可能，这里我们简要地提出一种。在某些情况下需要把数据拟合成指数函数。这种情况下的拟合是：

$$y = b(10)^{mx}$$

其中 x 是独立变量而 y 是因变量。现在我们定义关系：

$$w = \log_{10} y$$

$$z = x$$

然后用下面的形式进行数据拟合：

$$w = p_1 z + p_2$$

其中的系数 p_1 和 p_2 通过调用 *polyfit* 产生。这种拟合可以在 **MATLAB** 使用下面的命令产生：

```
p = polyfit(x, log10(y), 1)
```

我们可以求出 m 和 b：

$$m = p_1 \ , \ b = 10^{p_2}$$

习题

考虑下面的数据，一位奥林匹克举重教练已经收集了每个举重运动员的年龄和最大举重磅数，他相信这两者之间存在着函数关系。



年龄	最大举重
15	330
17	370
18	405
19	420
24	550
30	580
35	600
37	580

1. 使用 **MATLAB** 求出数据一次拟合的系数 m 和 b。
2. 创建一个年龄数组，以便估算当前队伍中队员（要求每一岁都有）的最大举重。
3. 创建一个函数 y 实现一次拟合。
4. 这个模型预计 17 岁的队员最大举重为多少？
5. 实际的举重均值为多少？
6. 这个模型预计的举重均值为多少？
7. 为了计算 r^2，请先求出 S 和 A。
8. 本模型的 r^2 是多少？
9. 尝试求二次拟合，这个函数是什么？
10. 二次拟合的 r^2 值是多少？

【参考答案在第 242 页】

第十一章



使用特殊函数

在数学、物理学和工程的应用上出现一些称为**特殊函数**的“不祥之物”。像**贝塞耳函数**（*Bessel Functions*）和**球谐函数**（*Spherical Harmonics*）等“禽兽”就是这些“不祥之物”的一员。本章我们将讨论如何使用 **MATLAB** 与这些函数一起工作。



Γ(伽马)函数

伽马函数使用$\Gamma(z)$表示，使用下面的积分定义：

$$\Gamma(n) = \int_0^\infty e^{-t} t^{n-1} dt$$

可以看出$\Gamma(1)=1$，这可以通过直接进行积分求得：

$$\Gamma(1) = \int_0^\infty e^{-t} t^{1-1} dt = \int_0^\infty e^{-t} dt = -e^{-t}\Big|_0^\infty = 1$$

如果进行部分积分，你会发现伽马函数遵从递归的关系：

$$\Gamma(n+1) = n\Gamma(n)$$

当n为正整数时，伽马函数将转换成阶乘函数：

$$\Gamma(n+1) = n!$$

使用积分计算我们还能得到：

$$\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$$

当$0 < x < 1$时，伽马函数满足：

$$\Gamma(x)\Gamma(1-x) = \frac{\pi}{\sin \pi x}$$

当n比较大时，递归函数由斯特灵公式(*Stirling's Formula*)近似给出：

$$n! \sim \sqrt{2\pi n}\; n^n e^{-n}$$

最后一点，伽马函数可以用来定义欧拉常数(*Euler's constant*)，即：

$$\Gamma'(1) = \int_0^\infty e^{-t} \ln t\, dt = -\gamma = 0.577215...$$

MATLAB 中的伽马函数

在**MATLAB**中，n的伽马函数可以使用下面的形式访问：

```
x = gamma(n)
```

例如，$\Gamma(6) = 5! = 120$，在**MATLAB**检验它：

```
>> gamma(6)
ans =
   120
```

因此在**MATLAB**使用伽马函数来确定阶乘的值是很简单的。让我们绘制伽马函数的图象，首先我们定义计算区间，绘制这个区间上所有实数值的伽马函数的图象，区间定义为：

```
>> n = linspace(0,5);
```

我们使用*plot*(*x*, *y*)产生图象：

```
>> plot(n,gamma(n)),xlabel('n'),ylabel('Gamma(n)'),grid on
```

结果如图11-1所示。看来当$n\to 0$时$\Gamma(n)$会急速上升，同时当$n>5$时也会快速增长，你可以通过计算函数的阶乘值认识到这一点。

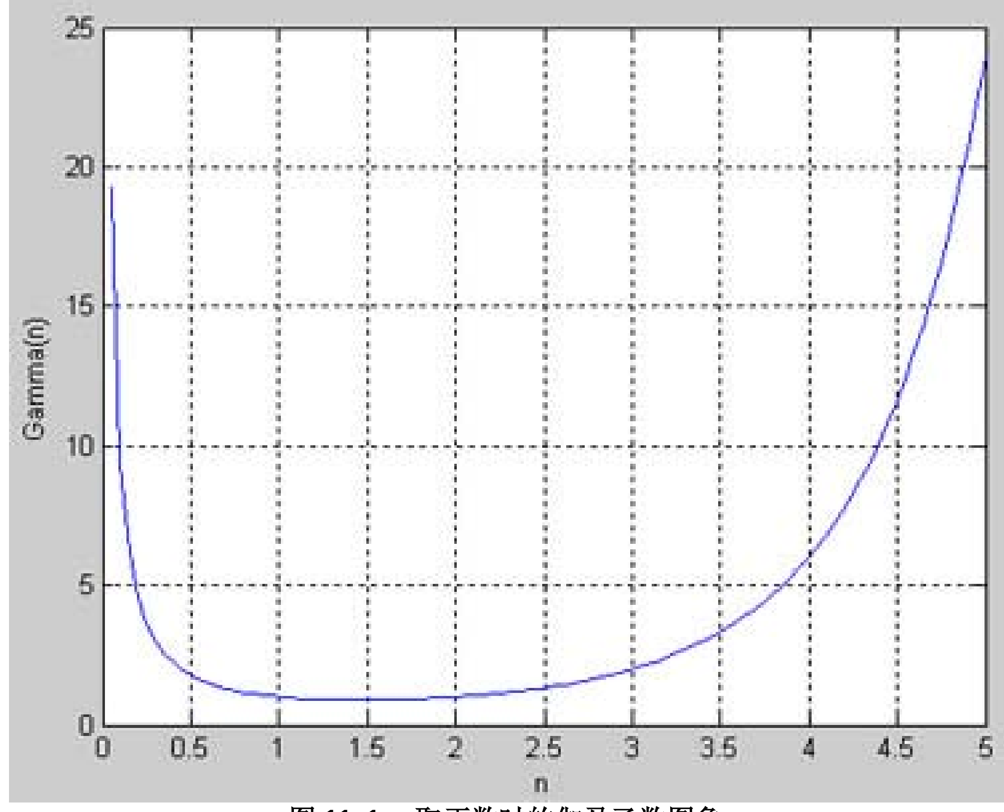


图 11-1 n取正数时的伽马函数图象

现在让我们看看*n*取负值时伽马函数的特点。首先让我们尝试某些整数的情况：

```
>> y = gamma(-1)
y =
   Inf
>> y = gamma(-2)
y =
   Inf
```

情况不妙，看来伽马函数好像对负参数都是得到无穷大值。不过如果我们尝试其它一些数值，就会看到情况并不总是这样。事实上就像是在正值与负值之间振荡一样：

```
>> y = gamma(-0.5)
y =
   -3.5449
>> y = gamma(-1.2)
y =
    4.8510
>> y = gamma(-2.3)
y =
   -1.4471
```

让我们绘制函数取负值时的图象。我们发现负整数值都是$\Gamma(x)$急剧增大的渐近线，如图11-2所示。

让我们把伽马函数的值列成表，我们列出$\Gamma(x)$在1.00到2.00之间步长为0.1的值。首先我们定义x值，要以表格显示这些数据，记得要像下面一样在行末包含单引号（可以试一下不含单引号的情况，看看**MATLAB**会如何显示数据）：

```
>> x = (1:0.1:2)';
```



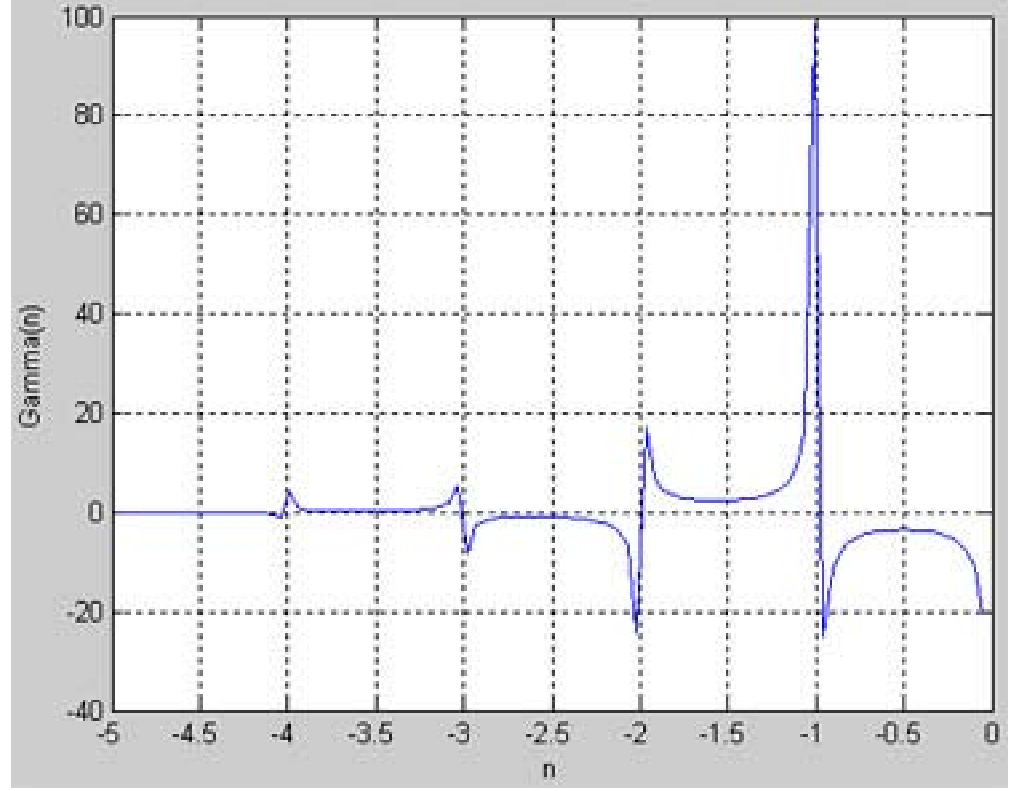


图11-2 MATLAB绘制的伽马函数取负值时的图象

现在我们定义一个数组用来保存伽马函数的值：

```
>> y = gamma(x);
```

我们可以使用这些数值创建一个矩阵并把这些数值用表格列出来：

```
>> A = [x y]
A =
    1.0000    1.0000
    1.1000    0.9514
    1.2000    0.9182
    1.3000    0.8975
    1.4000    0.8873
    1.5000    0.8862
    1.6000    0.8935
    1.7000    0.9086
    1.8000    0.9314
    1.9000    0.9618
    2.0000    1.0000
```

例11-1

使用伽马函数计算任意维数的球体的表面积。设球体的半径$r = 1$，并考虑维数$n = 2$，3和11的情况。

解11-1

设任意球体从原点量得的距离或半径为r。n维球体的表面积由下式给出：

$$S_n = r^{n-1}\frac{2\pi^{n/2}}{\Gamma\left(\frac{n}{2}\right)}$$

我们可以在**MATLAB**中创建一个内联函数来计算球体的表面积：

```
>> surface = @(n,r) r^(n-1)*2*(pi^(n/2))/gamma(n/2)
```

我们先做一些快速的手工计算以确保这个公式是正确的。对于二维的情况，这个“球体”就是一个圆，而它的表面积就是它的周长：

$$S_2 = r^{2-1}\frac{2\pi^{2/2}}{\Gamma\left(\frac{2}{2}\right)} = r\frac{2\pi}{\Gamma(1)} = 2\pi r$$

由于$\Gamma(1) = 1$。如果我们令$r = 1$，那表面积是：

$$S_2 = 2\pi = 6.2832$$

调用我们的函数：

```
>> surface(2,1)
ans =
    6.2832
```

现在让我们转移到三维的情况。一个球体的表面积是：

$$S_3 = r^{3-1}\frac{2\pi^{3/2}}{\Gamma\left(\frac{3}{2}\right)} = r^2\frac{2\pi^{3/2}}{\frac{1}{2}\Gamma\left(\frac{1}{2}\right)} = \frac{4\pi^{3/2}r^2}{\sqrt{\pi}} = 4\pi r^2$$

如果$r = 1$，那么表面积等于$4\pi = 12.5664$。检查我们的函数：

```
>> surface(3,1)
ans =
   12.5664
```

因此那个字串（公式S_n）能够满足那些理论家的白日梦，让我们检查$n = 11$的情况：

$$S_{11} = r^{11-1}\frac{2\pi^{11/2}}{\Gamma\left(\frac{11}{2}\right)} = \frac{2\pi^{11/2}\ r^{10}}{\left(\frac{945}{32}\right)\Gamma\left(\frac{1}{2}\right)} = \frac{4\pi^{11/2}r^{10}}{\left(\frac{945}{32}\right)\sqrt{\pi}} = \frac{64}{945}\pi^{11}r^{10}$$

我们的函数求得：

```
>> surface(11,1)
ans =
   20.7251
```

与伽马函数相关的数

MATLAB允许你计算**不完全伽马函数(*incomplete gamma function*)**，它的定义如下：

$$p(x,\ n) = \frac{1}{\Gamma(n)}\int_0^x e^{-t}t^{n-1}dt$$

MATLAB中用来求这个函数的命令是：

```
y = gammainc(x,n)
```

当$x<<1$和$n<<1$时，不完全伽马函数满足$p(x,\ n) \approx x^n$。我们可以通过一些计算检验它：

```
>> x = 0.2;
>> n = 0.3;
>> y = x^n
y =
    0.6170
>> z = gammainc(x,n)
```

```
z =
    0.6575
```

取更小的数值时，这种近似更加接近：

```
>> x = 0.002; n = 0.003; y = x^n
y =
    0.9815
>> z = gammainc(x,n)
z =
    0.9832
```

贝塞耳函数

贝塞耳微分方程(*Bessel's differential equation*)在很多工程和科学上都有应用。这个方程的形式是：

$$x^2y'' + xy' + (x^2 - n^2)y = 0$$

这个方程的解是：

$$y(x) = A_1J_n(x) + A_2Y_n(x)$$

其中A_1和A_2由边界条件确定的常数。解中的函数$J_n(x)$就是**第一类贝塞耳函数**(*Bessel function of the first kind*)而$Y_n(x)$是**第二类贝塞耳函数**(*Bessel function of the second kind*)或者是**诺埃曼函数**(*Neumann function*)。

第一类贝塞耳函数用下面的无穷级数定义，伽马函数是定义的一部分：

$$J_n(x) = \sum_{k=0}^{\infty}\frac{(-1)^k(x/2)^{n+2k}}{k!\Gamma(n + k + 1)}$$

在**MATLAB**中，第一类贝塞耳函数使用*besselj*实现。调用的形式是：

```
y = besselj(n,x)
```

要对第一类贝塞耳函数的特点有些印象，我们绘制它的一些图象，我们可以使用下面的命令产生$J_1(x)$的图象：

```
>> x = [0:0.1:50]; y = besselj(1,x);
>> plot(x,y),xlabel('x'),ylabel('BesselJ(1,x)')
```

结果如图11-3所示。注意当$x\to0$是，$J_1(x)$是有限的，趋于0。我们还可以看到这个函数具有衰减振荡的动作特点。

这些特征普遍适用于第一类贝塞耳函数。让我们产生一个图象比较$J_0(x)$、$J_1(x)$和$J_2(x)$：

```
>> x = [0:0.1:20]; u = besselj(0,x); v = besselj(1,x); w = besselj(2,x);
>> plot(x,u,x,v,'--',x,w,'-.'),xlabel('x'),ylabel('BesselJ(n,x)'),...
grid on, legend('bessel0(x)','bessel1(x)','bessel2(x)')
```

看图11-4中的图象，可以看出，随着n增大振荡越来越小。

第一类贝塞耳函数也定义在负整数上。它们与第一类贝塞耳函数的在正整数上的定义相关：

$$J_{-n}(x) = (-1)^nJ_n(x)$$

我们做一些符号计算，求贝塞耳函数的导数公式：

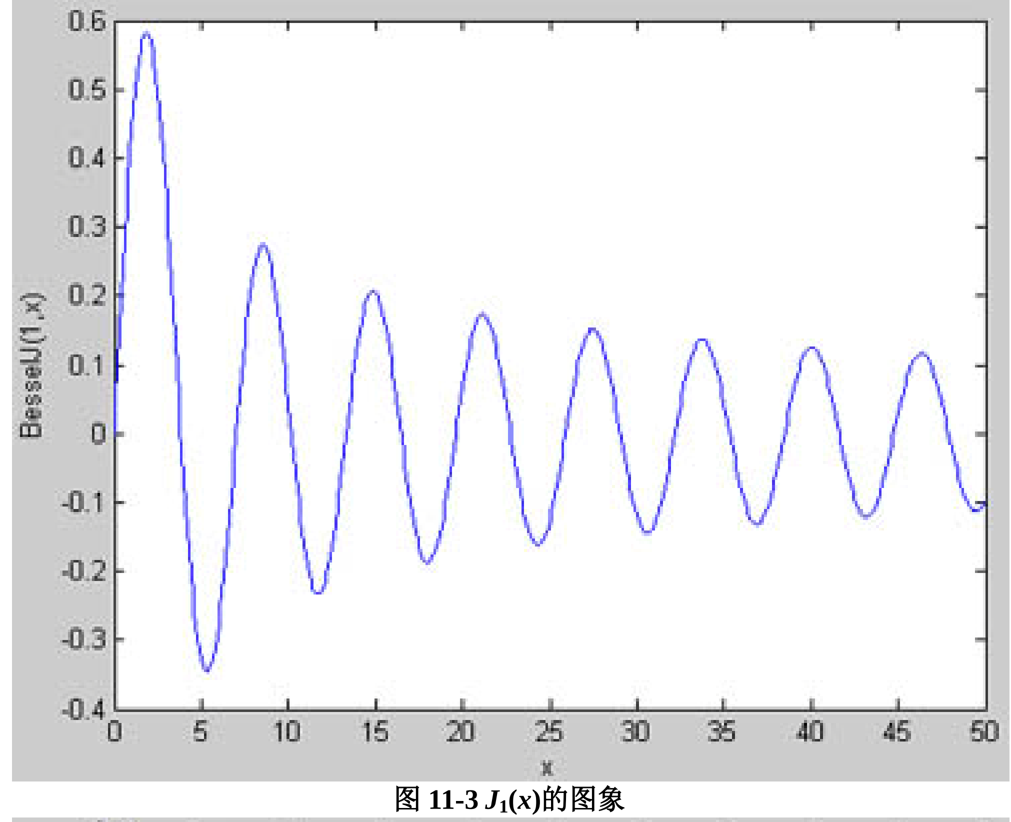


图 11-3 $J_1(x)$的图象

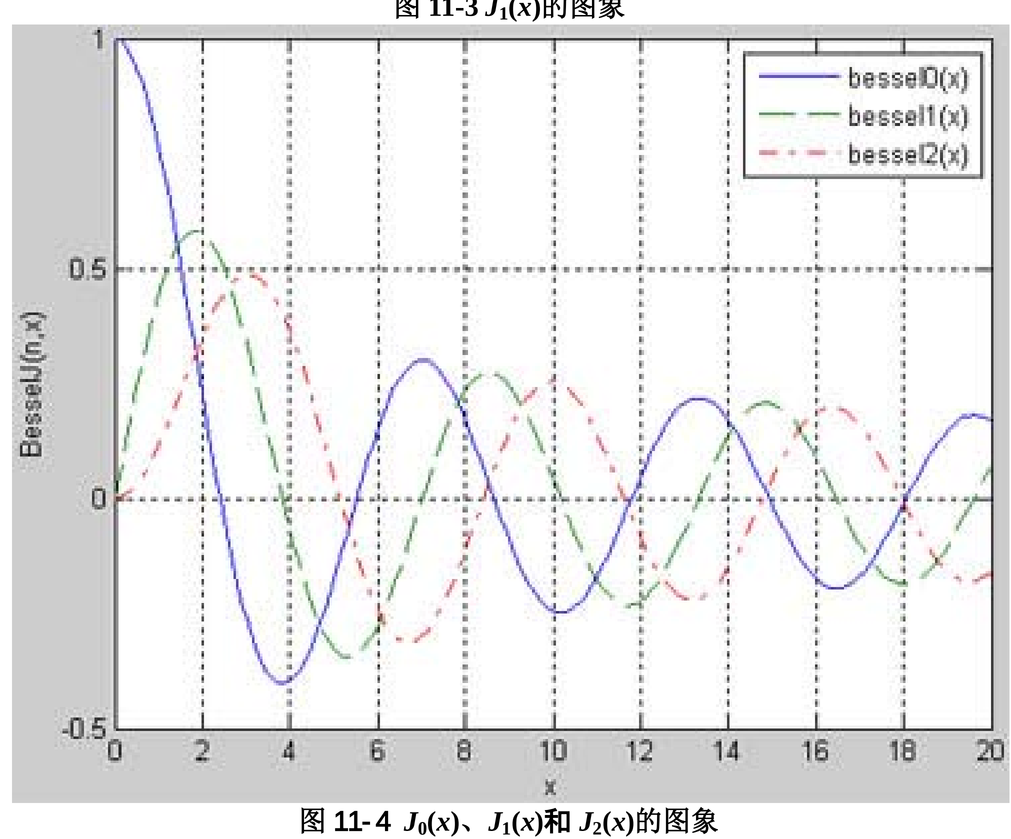


图 11-4 $J_0(x)$、$J_1(x)$和 $J_2(x)$的图象

```
>> syms n x y
>> diff(besselj(n,x))
ans =
-besselj(n+1,x)+n/x*besselj(n,x)
```



MATLAB已经计算出它们之间的关系：

$$J'_n(x) = \frac{n}{x}J_n(x) - J_{n+1}(x)$$

当遇到被积函数含有贝塞耳函数时，**MATLAB**也能够使用。例如：

```
>> syms x n;
>> int(x^n*besselj(n-1,x))
ans =
x^n*besselj(n,x)
```

因此，对积分常数求模：

$$\int x^n J_{n-1}(x)dx = x^n J_n(x)$$

让我们计算$J_0(x)$和$J_1(x)$的泰勒级数展开式的前五项：

```
>> taylor(besselj(0,x),5)
ans =
1-1/4*x^2+1/64*x^4
>> taylor(besselj(1,x),5)
ans =
1/2*x-1/16*x^3
```

MATLAB还内置了其它贝塞耳函数。第二类贝塞耳函数使用*bessely(n, x)*实现。在图11-5中，我们使用**MATLAB**产生前三个第二类贝塞耳函数的图象。

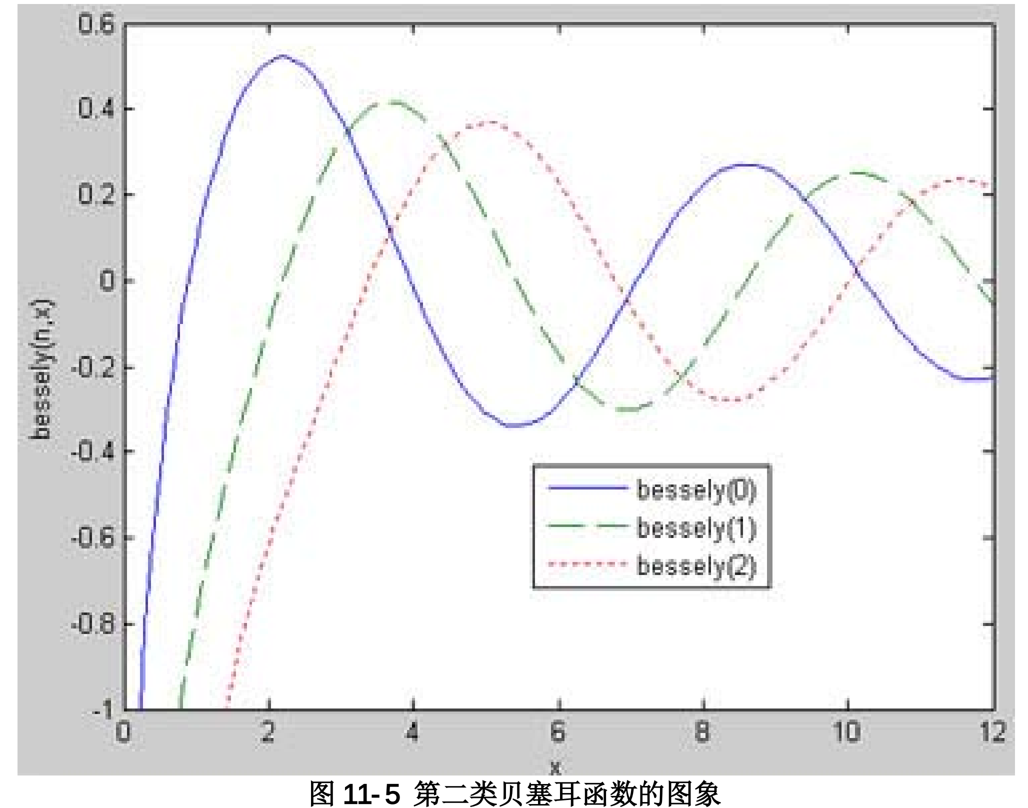


图 11-5 第二类贝塞耳函数的图象

我们还能够在**MATLAB**中实现其它类型的贝塞耳函数——**汉克尔函数**(*Hankel Function*)。调用*besselh(nu, k, z)*即可利用这些函数，一共有两类的汉克尔函数（第一类和第二类），在**MATLAB**中函数的类型由*k*指出。如果我们把*k*从参数中省略而写成*besselh(nu, z)*，**MATLAB**默认是使用第一类汉克尔函数。

例11-2

柱面行波可以使用下面的公式来描述：

$$\Psi(r,t) = H_0^{(1)}(kr)e^{-i\omega t}$$

我们取其中的实数部份。在**MATLAB**中实现这个函数的实数部分并把它看成是r(单位为m)的函数，绘制$t=0$，$\pi/2\omega$，$3\pi/4\omega$时的图象。设在$0\leqslant r\leqslant 10$m范围内$k = 500\text{cm}^{-1}$。

解11-2

首先我们把k转成合适的单位：

$$k = 500\text{cm}^{-1}\left(\frac{100\text{cm}}{\text{m}}\right) = 50000\text{m}^{-1}$$

让我们产生一些图象看看k如何对汉克尔函数的实数部分产生影响，首先我们为图象的绘制区间创建数组：

```
>> r = linspace(0,10);
```

假设k是一致的，这样我们就可以定义汉克尔函数为：

```
>> u = besselh(0,r);
```

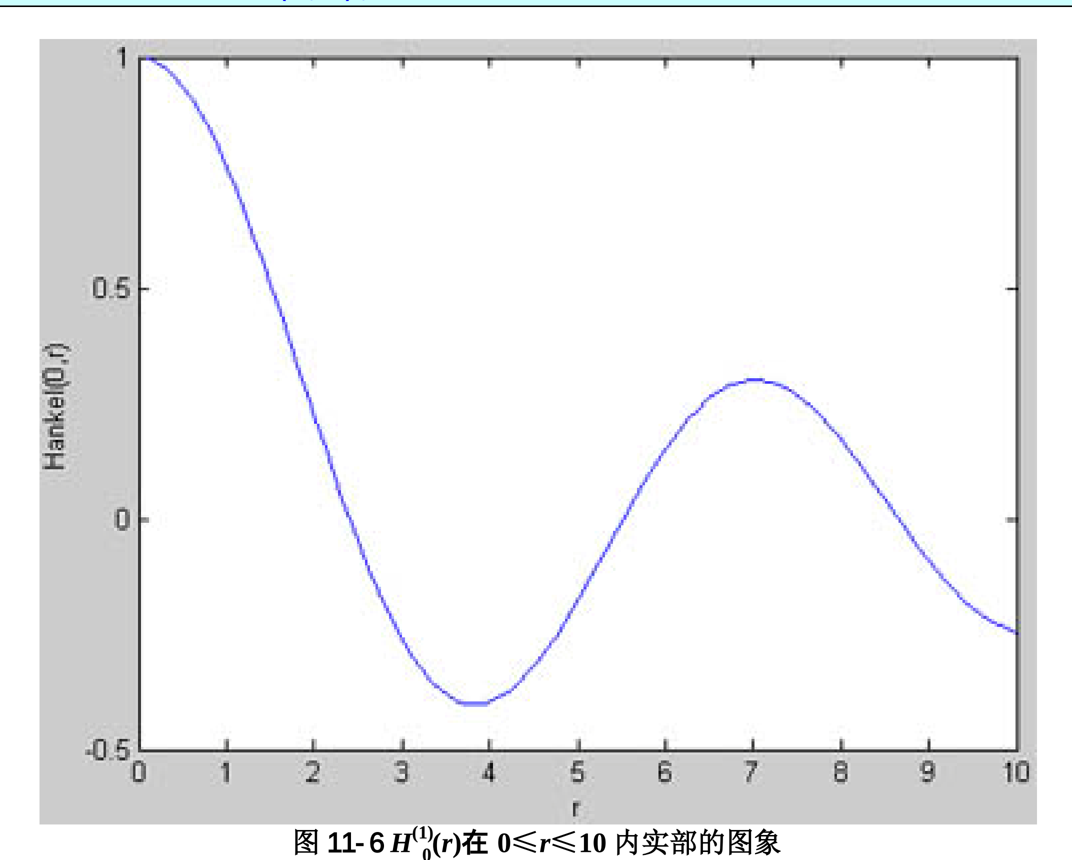


图 11-6 $H_0^{(1)}(r)$在 $0\leqslant r\leqslant 10$ 内实部的图象

计算不同点上的汉克尔函数值你会发现它们是复数。例如：

```
>> besselh(0,2)
ans =
   0.2239 + 0.5104i
```

这表明对于我们所讨论的物理上的函数，我们需要计算它的实部，在学习电磁行波时一个常见的练习就是计算实部。因此让我们绘制这个函数实部的图象：

```
>> plot(r,real(u)),xlabel('r'),ylabel('Hankel(0,r)')
```

如图11-6所示，结果表明这个函数是一个衰减振荡函数，这就难怪了，因为它是一种贝塞耳函数，而我们已经知道贝塞耳函数都是衰减振荡函数！



现在我们把波数k添加进去，上面的代码变成：

```
>> r = linspace(0,10);
>> k = 50000;
>> w = besselh(0,k*r);
>> plot(r,real(w)),xlabel('r'),ylabel('Hankel(0,kr)')
```

观察图11-7中的图象，我们可以发现两点：第一，函数在距离r上快速振荡；第二，本例中振幅非常小。

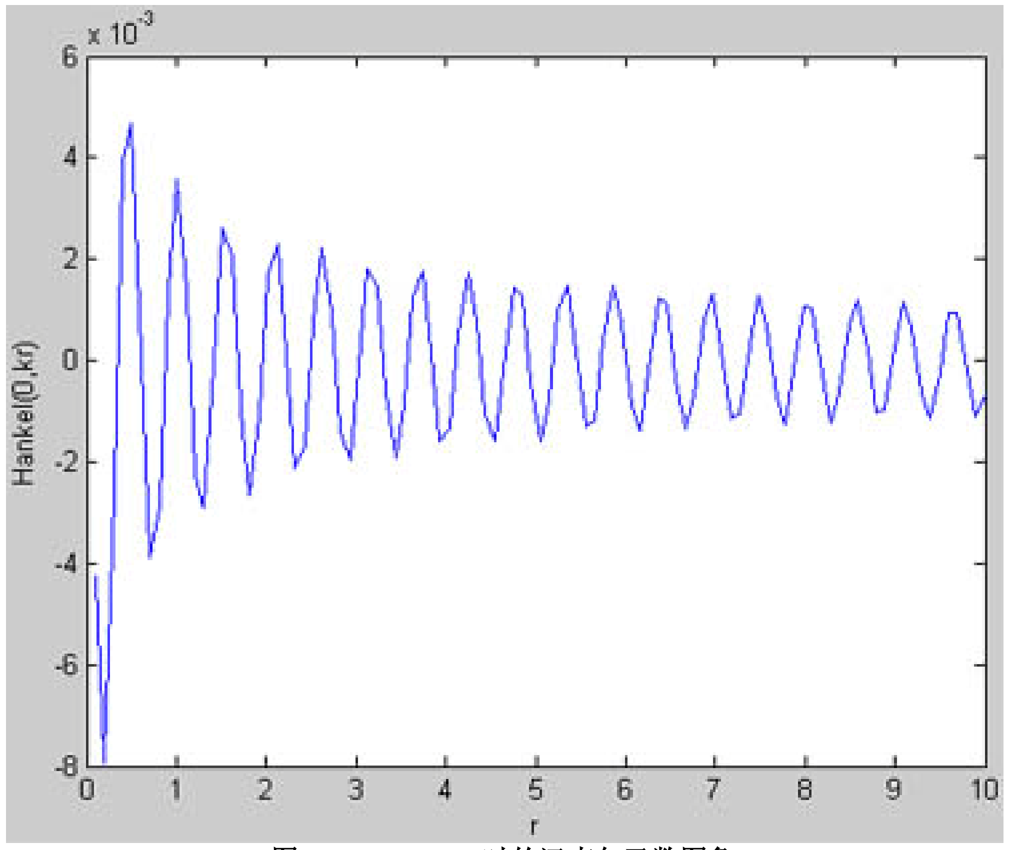


图 11-7 k=50,000 时的汉克尔函数图象

现在我们看看在其它时间内这个函数的情况。首先考虑$t = 0$时，有：

$$\Psi(r, 0) = H_0^{(1)}(kr)$$

这时的图象就是我们在图11-7画出来的。现在我们令$t = \pi/2\omega$：

$$\Psi(r, \pi/2\omega) = H_0^{(1)}(kr)e^{-i\omega(\pi/2\omega)} = H_0^{(1)}(kr)e^{-i\pi/2} = -iH_0^{(1)}(kr)$$

要计算结果我们需要用到欧拉公式(*Euler's formula*)：

$$e^{i\theta} = \cos\theta + i\sin\theta$$

在图11-8中，我们把这个函数实部的图象(虚线)跟$t = 0$时的图象画在一起。

看起来波形有点移位而已，并没有太大不同。这函数描述了一个在r增大的方向上的衰减波。

最后，我们创建函数：

```
>> s = besselh(0,k*r)*(cos(3*pi/4)-i*sin(3*pi/4));
```

它的图象如图11-9所示。

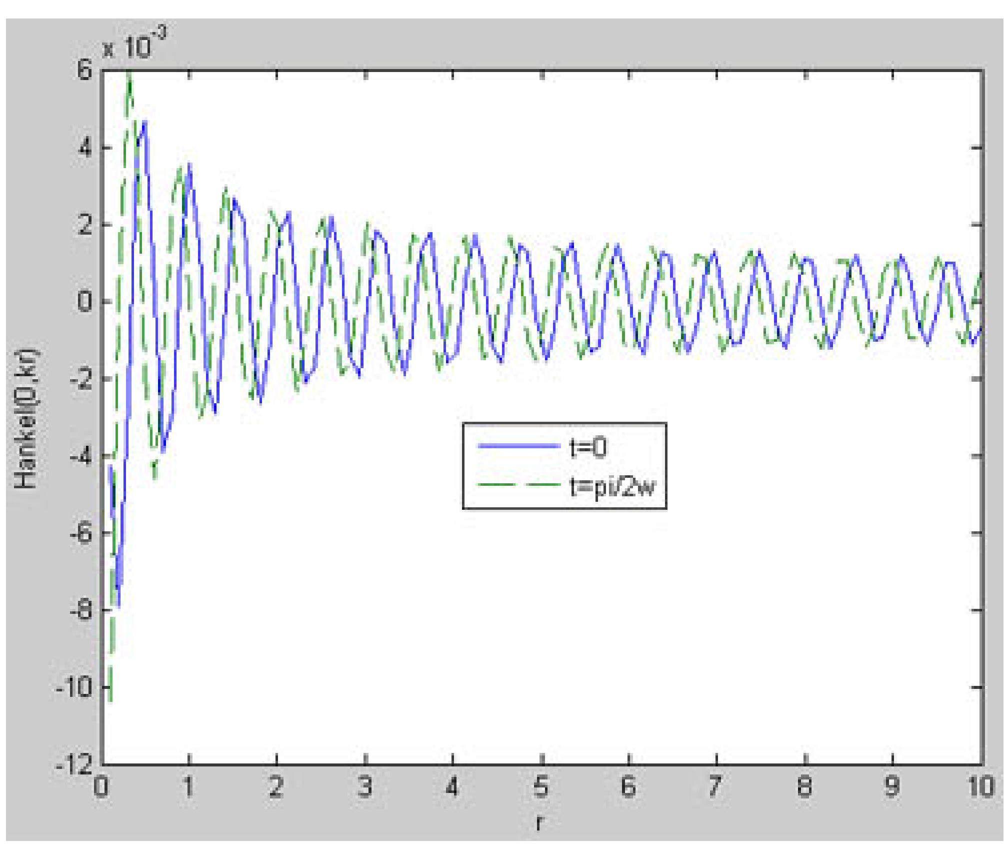


图 11-8 函数在 $t = 0$ 和 $t = \pi/2\omega$ 时的图象

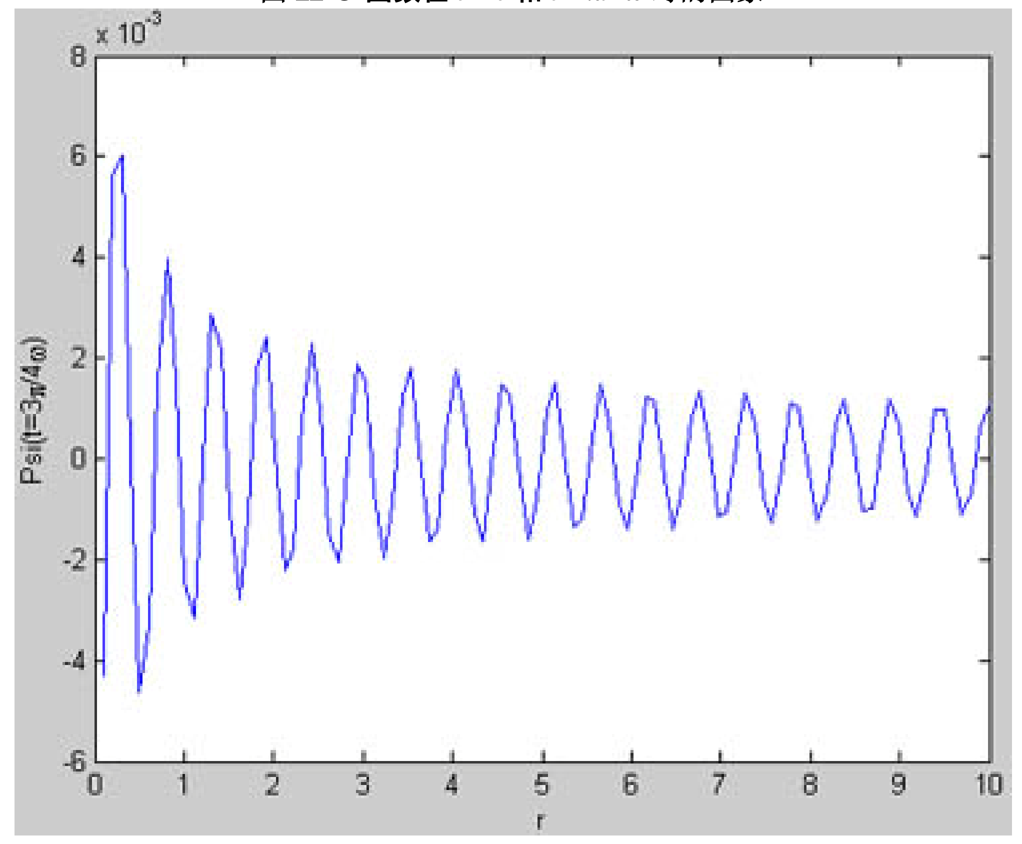


图 11-9 $\Psi(r, t) = H_0^{(1)}(kr)e^{-i\omega t}$在 $t=3\pi/4\omega$ 时的实部图象

检查汉克尔函数，表明它不能计算在0点时的值：

```
>> besselh(0,0)
```

```
ans =
     NaN +    NaNi
```

MATLAB 使用NaN来表示“不是数值(*not a number*)”

贝塔函数

贝塔函数(*Beta function*)带有两个参数，采用下面的积分定义：

$$B(m,n) = \int_0^1 x^{m-1}(1-x)^{n-1}dx$$

如果$m, n > 0$则积分收敛。要在**MATLAB**中使用贝塔函数，我们用：

```
x = beta(m,n)
```

我们可以使用**MATLAB**产生贝塔函数的数表。如果$m = 1$，我们发现$beta(1,n)$的前十个值是：

```
>> x = (1:1:10)'; y = beta(1,x); A = [x y]
A =
    1.0000    1.0000
    2.0000    0.5000
    3.0000    0.3333
    4.0000    0.2500
    5.0000    0.2000
    6.0000    0.1667
    7.0000    0.1429
    8.0000    0.1250
    9.0000    0.1111
   10.0000    0.1000
```

贝塔函数有什么用呢？研究表明贝塔函数可以用来计算含有$\sin\theta$和$\cos\theta$乘积的积分。

例11-3

求下式积分值。

$$\int_0^{\pi/2}\sin^5\theta\cos^3\theta d\theta$$

解11-3

要求这个积分，我们需要挖掘微积分书上的一些技巧，否则将处于无穷尽的分部积分计算之中。其实我们可以使用下面的关系式：

$$\int_0^{\pi/2}\sin^{2m-1}\theta\cos^{2n-1}\theta d\theta = \frac{1}{2}B(m,n)$$

因此：

$$\int_0^{\pi/2}\sin^5\theta\cos^3\theta d\theta = \frac{1}{2}B(3,2)$$

使用**MATLAB**求得答案是：

```
>> (0.5)*beta(3,2)
ans =
    0.0417
```

$$\Rightarrow \int_0^{\pi/2}\sin^5\theta\cos^3\theta d\theta = 0.0417$$

例11-4

求

$$\int_0^{\pi/2} \sin^{19}\theta d\theta$$

解11-4
这次$m = 10$, $n = 1/2$，因此这个积分等于：

```
>> (1/2)*beta(10,1/2)
ans =
    0.2838
```

例11-5
计算

$$\int_0^{\pi/2} \sqrt{\tan\theta}\, d\theta$$

解11-5
经过一些小处理可以将它变成可以应用贝塔函数的形式：

$$\int_0^{\pi/2} \sqrt{\tan\theta}\, d\theta = \int_0^{\pi/2} \sqrt{\frac{\sin\theta}{\cos\theta}}\, d\theta = \int_0^{\pi/2} \sin^{1/2}\theta\cos^{-1/2}\theta d\theta$$

因此我们有：

$$2m - 1 = 1/2, \Rightarrow m = 3/4$$
$$2n - 1 = -1/2, \Rightarrow n = 1/4$$

因此此积分计算等于$1/2B(3/4, 1/4)$的值，具体为：

```
>> 0.5*beta(3/4,1/4)
ans =
    2.2214
```

这刚好等于$\pi/\sqrt{2}$

特殊积分

在数学物理学中有很多“特殊积分”。本节中，我们将提到一些，并学习如何在**MATLAB**用它们来计算。第一我们考虑的是**幂积分**(*exponential integral*)：

$$E_i(x) = \int_x^{\infty} \frac{e^{-u}}{u} du$$

在**MATLAB**中使用下面的语法来执行这个函数：

```
y = expint(x)
```

这里我们把数值列成表。注意$expint(0) = inf$。

```
>> x = (0.1:0.1:2)'; y = expint(x); A = [x y]
A =
    0.1000    1.8229
    0.2000    1.2227
    0.3000    0.9057
    0.4000    0.7024
    0.5000    0.5598
    0.6000    0.4544
    0.7000    0.3738
    0.8000    0.3106
    0.9000    0.2602
    1.0000    0.2194
```

```
    1.1000    0.1860
    1.2000    0.1584
    1.3000    0.1355
    1.4000    0.1162
    1.5000    0.1000
    1.6000    0.0863
    1.7000    0.0747
    1.8000    0.0647
    1.9000    0.0562
    2.0000    0.0489
```

很多其它的特殊函数可以通过使用mfun命令进行数值计算。通过查询**MATLAB**的帮助，我们可以看到一个可以用来进行数值计算的函数表格：

```
>> help mfunlist
 MFUNLIST Special functions for MFUN.
    The following special functions are listed in alphabetical order
    according to the third column. n denotes an integer argument,
    x denotes a real argument, and z denotes a complex argument. For
    more detailed descriptions of the functions, including any
    argument restrictions, see the Reference Manual, or use MHELP.
```

```
 bernoulli        n          Bernoulli Numbers
 bernoulli        n,z        Bernoulli Polynomials
 BesselI          x1,x       Bessel Function of the First Kind
 BesselJ          x1,x       Bessel Function of the First Kind
 BesselK          x1,x       Bessel Function of the Second Kind
 BesselY          x1,x       Bessel Function of the Second Kind
 Beta             z1,z2      Beta Function
 binomial         x1,x2      Binomial Coefficients
 EllipticF -      z,k        Incomplete Elliptic Integral, First Kind
 EllipticK -      k          Complete Elliptic Integral, First Kind
 EllipticCK -     k          Complementary Complete Integral, First Kind
 EllipticE -      k          Complete Elliptic Integrals, Second Kind
 EllipticE -      z,k        Incomplete Elliptic Integrals, Second Kind
 EllipticCE -     k          Complementary Complete Elliptic Integral, Second Kind
 EllipticPi -     nu,k       Complete Elliptic Integrals, Third Kind
 EllipticPi -     z,nu,k     Incomplete Elliptic Integrals, Third Kind
 EllipticCPi -    nu,k       Complementary Complete Elliptic Integral, Third Kind
 erfc             z          Complementary Error Function
 erfc             n,z        Complementary Error Function's Iterated Integrals
 Ci               z          Cosine Integral
 dawson           x          Dawson's Integral
 Psi              z          Digamma Function
 dilog            x          Dilogarithm Integral
 erf              z          Error Function
 euler            n          Euler Numbers
 euler            n,z        Euler Polynomials
 Ei               x          Exponential Integral
 Ei               n,z        Exponential Integral
 FresnelC         x          Fresnel Cosine Integral
 FresnelS         x          Fresnel Sine Integral
 GAMMA            z          Gamma Function
 harmonic         n          Harmonic Function
 Chi              z          Hyperbolic Cosine Integral
 Shi              z          Hyperbolic Sine Integral
 GAMMA            z1,z2      Incomplete Gamma Function
 W                z          Lambert's W Function
 W                n,z        Lambert's W Function
 lnGAMMA          z          Logarithm of the Gamma function
 Li               x          Logarithmic Integral
 Psi              n,z        Polygamma Function
 Ssi              z          Shifted Sine Integral
 Si               z          Sine Integral
 Zeta             z          (Riemann) Zeta Function
 Zeta             n,z        (Riemann) Zeta Function
 Zeta             n,z,x      (Riemann) Zeta Function

      Orthogonal Polynomials (Extended Symbolic Math Toolbox only)
 T               n,x         Chebyshev of the First Kind
 U               n,x         Chebyshev of the Second Kind
 G               n,x1,x      Gegenbauer
 H               n,x         Hermite
 P               n,x1,x2,x   Jacobi
 L               n,x         Laguerre
```

```
    L              n,x1,x      Generalized Laguerre
    P              n,x         Legendre

       See also mfun, mhelp.


       Reference page in Help browser
          doc mfunlist
```

为了了解如何使用这个函数，我们考虑**黎曼ζ函数**(*Riemann zeta function*)，它由下面的级数定义：

$$\zeta(z) = \sum_{n=1}^{\infty}\frac{1}{n^z}$$

参数z是满足$Re(z)>1$的复数，这个ζ函数在实整数上的计算会产生一些有趣的结果，这可以通过前面几个数值的计算结果看出。当$z = 1$时，我们得到趋于无穷大的**调和级数**(*harmonic series*)：

$$\zeta(1) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots = \infty$$

继续的话，我们得到一些有趣的关系式：

$$\zeta(2) = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots = \frac{\pi^2}{6} = 1.645$$

$$\zeta(3) = 1 + \frac{1}{2^3} + \frac{1}{3^3} + \ldots = 1.202$$

事实上只要参数是偶数，即m是一个偶数，那么ζ函数总是与下式成比例：

$$\zeta(m)\bullet\pi^m$$

要在**MATLAB**使用黎曼ζ函数计算，我们写成：

```
w = mfun('Zeta',z)
```

例如，通过计算ζ(2)的值我们即可得到π的大小：

```
>> w = mfun('Zeta',2)
w =
    1.6449
>> my_pi = sqrt(6*w)
my_pi =
    3.1416
```

让我们绘制ζ函数在正实数上的图象。首先我们定义区间：

```
>> x = linspace(0,10);
```

接着我们定义ζ函数并绘制它：

```
>> w = mfun('Zeta',x);
>> plot(x,w),xlabel('x'),title('Zeta(x)'),axis([0 10 -10 10])
```

图象如图11-10所示。正如我们前面所看到的，ζ函数在$x=1$时趋于无穷大，这在图象中被显示出来。在$x > 1$时它快速衰减，通过使用一些数值进行检验，我们可知这函数收敛于1：

```
>> w = mfun('Zeta',1000)
w =
     1
>> w = mfun('Zeta',1000000)
w =
     1
```



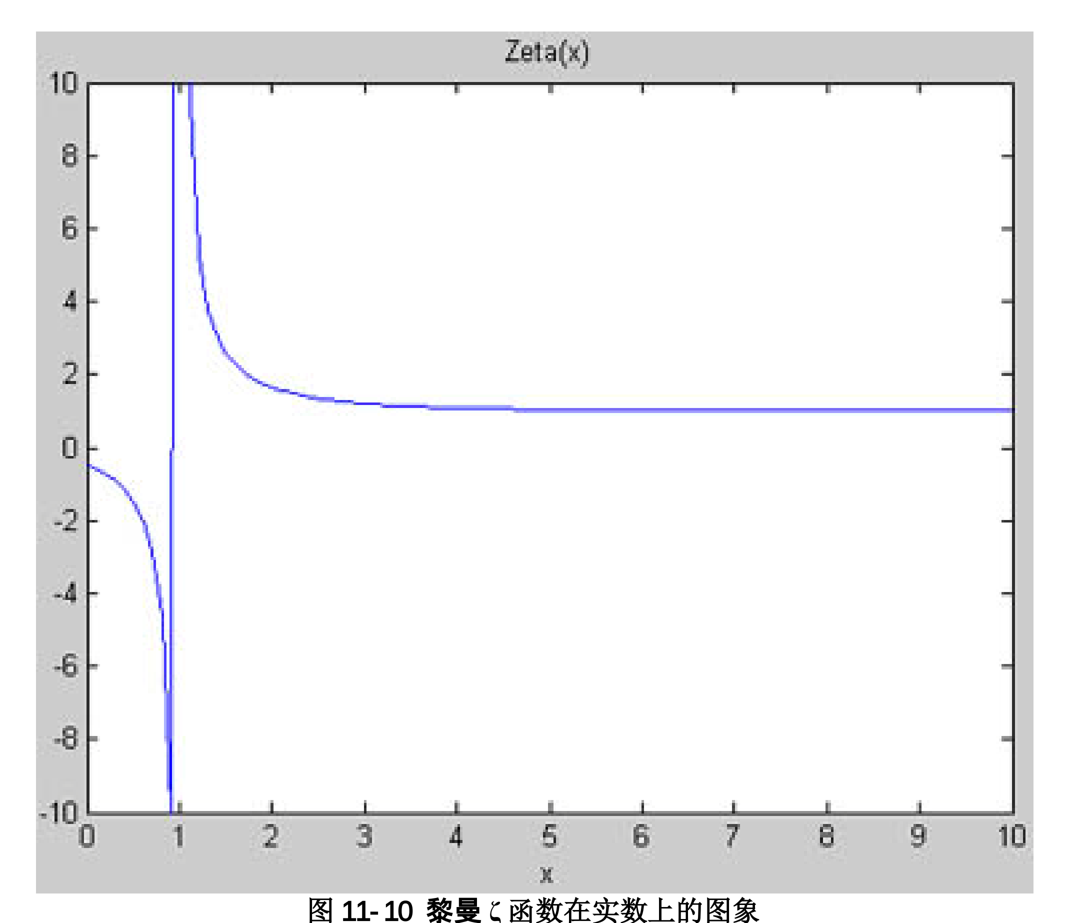


图 11-10 黎曼ζ函数在实数上的图象

勒让德函数

勒让德微分方程(*Legendre's differential equation*)由下式给出：

$$(1 - x^2)y'' - 2xy' + n(n + 1)y = 0$$

其中n是一个非负整数。这个方程的解可以写成：

$$y = a_1P_n(x) + a_2Q_n(x)$$

$P_n(x)$一个多项式，称为**勒让德多项式**(*Legendre polynomials*)。通过使用**罗德里格斯公式**(*Rodrigues' formula*)公式它们可以化为：

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n$$

相伴（连带？关联？）勒让德微分方程(*Legendre's associated differential equation*)由下式给出：

$$(1 - x^2)y'' - 2xy' + \left[n(n + 1) - \frac{m^2}{1-x^2}\right] y = 0$$

这里m是一个非负整数。这个方程的解具有下面的形式：

$$y = a_1P_n^m(x) + a_2Q_n^m(x)$$

这里$P_n^m(x)$是**第一类相伴勒让德方程**(*associated Legendre functions of the first kind*)而$Q_n^m(x)$是**第二类相伴勒让德方程**(*associated Legendre functions of the second kind*)。如果$m > n$则$P_n^m(x) = 0$。相伴勒让德方程在**MATLAB**中可以使用下面的命令来计算：

```
p = legendre(n,x)
```

相伴勒让德方程只有在$-1 \leqslant x \leqslant 1$内有效。函数*legendre*($n$, x)可以计算第一类相伴勒让德函数$P_n^0(x), P_n^2(x), ..., P_n^n(x)$。还可以用来进行相伴勒让德函数的数值计算。例

如，假设我们要计算n=1时的值，这相当于告诉我们$m = 0$或$m = 1$。首先我们在$-1 \leqslant x \leqslant 1$内取出间隔均匀的点：

```
>> x = (-1:0.5:1)
x =
   -1.0000   -0.5000         0    0.5000    1.0000
```

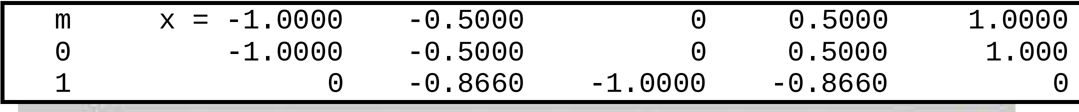
现在我们计算$n = 1$的函数值：

```
>> p = legendre(1,x)
```

数据以下面的形式显示出来：

```
p =
   -1.0000   -0.5000         0    0.5000    1.0000
         0   -0.8660   -1.0000   -0.8660         0
```

上面表中，行表示m，列表示x。我们把它放进表格中，以便明白是如何计算的：



m	x = -1.0000	-0.5000	0	0.5000	1.0000
0	-1.0000	-0.5000	0	0.5000	1.000
1	0	-0.8660	-1.0000	-0.8660	0

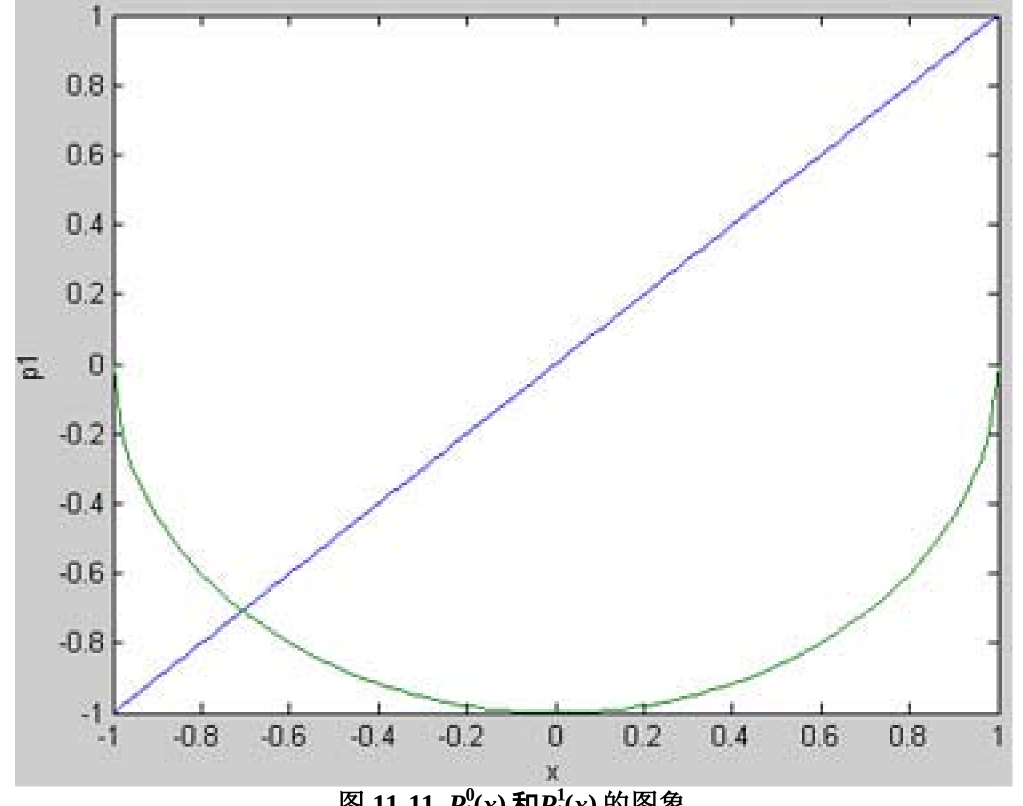


图 11-11 $P_1^0(x)$和$P_1^1(x)$的图象

因此，$P_1^0(-1) = -1$，$P_1^1(-0.5) = -0.866$。把这些函数绘制成图象会看到非常有趣的结果。在图11-11中，我们显示了legendre(1, x)的图象，它包括了$m = 0$和$m = 1$时的曲线，其中直线是$P_1^0(x)$，曲线是$P_1^1(x)$。用来产生图象的代码如下。首先我们定义区间：

```
>> x = linspace(-1,1);
```

接着我们建立相伴勒让德函数：



```
>> p1 = legendre(1,x); p2 = legendre(2,x); p3 = legendre(3,x);p4 = legendre(4,x);
```

要绘制第一个图象，我们输入：

```
>> plot(x,p1),xlabel('x'),ylabel('p1')
```

现在我们产生legendre(2, x)的图象，在这种情况下$n = 2$，意味着$m = 0, 1, 2$. 结果如图11-12所示。首先我们看看如何把每个函数提取出来，这可以通过使用**MATLAB**标准的数组定位方法做到。与我们前面看到的一样，legendre(n, x)每一行代表每一个m的函数值。在legendre(2, x)的情况中，第一行是$P_2^0(x)$，第二行是$P_2^1(x)$，第三行是$P_2^2(x)$。现在让我们把这些数据转换成独立的数组，使用$b = A(i, :)$就是告诉**MATLAB**我们要把第i行的所有列(由"："说明)取出来放进数组b中去。在本例中我们写成：

```
>> f = p2(1,:);
>> g = p2(2,:);
>> h = p2(3,:);
```

现在我们可以在同一个图绘制这三个函数的图象了，同时标记这三条曲线：

```
>> plot(x,f,x,g,'--',x,h,':'),xlabel('x'), ...
ylabel('p2'), legend('p(2,0)','p(2,1)','p(2,2)')
```

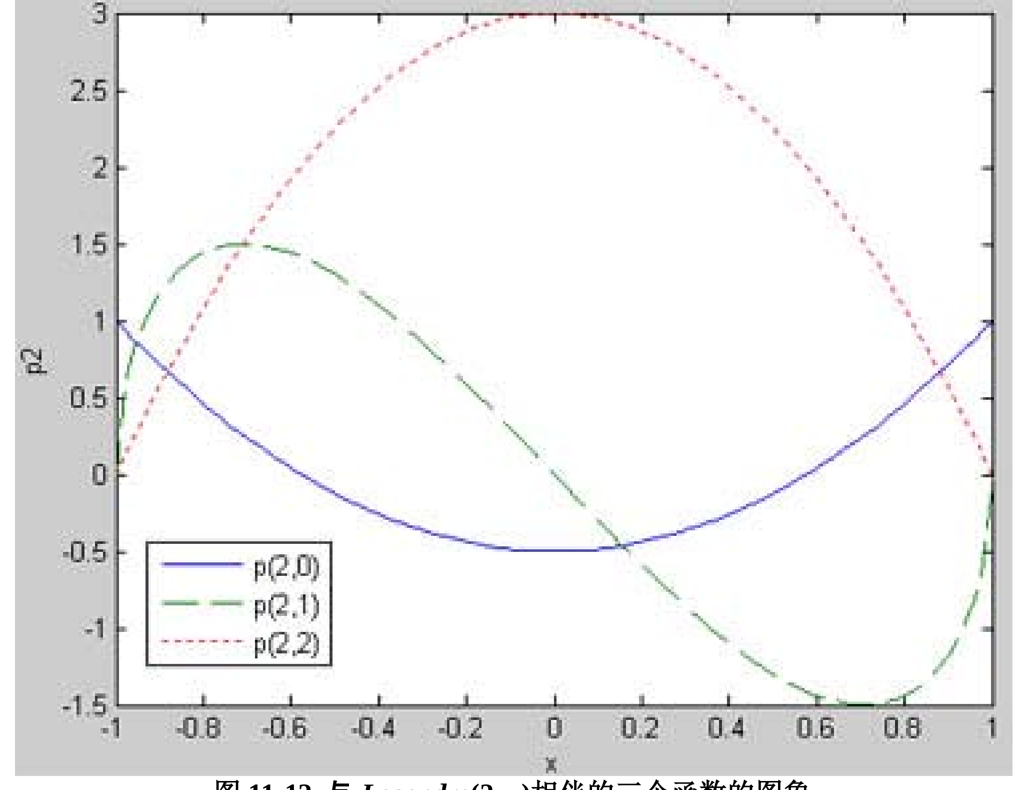


图 11-12 与 *Legendre*(2, *x*)相伴的三个函数的图象

亚里函数

以下形式的微分方程的解被称为**亚里函数**(**空想函数*Airy functions***)：

$$\frac{d^2\Psi}{dx^2} - x\Psi = 0$$

物理学上，学习量子力学时需要用到亚里函数。我们用$Ai(z)$来表示亚里函数。当亚里函数的参数为实数时，它可以写成：

$$Ai(x) = \frac{1}{\pi}\int_0^\infty \cos\left(\frac{t^3}{3} + xt\right)dt$$

在**MATLAB**中使用`w = airy(z)`来计算$Ai(z)$的值。对于实数参数，当$x = 0$时亚里函数可以写成伽马函数的形式：

$$Ai(0) = \frac{1}{3^{2/3}\Gamma(2/3)}$$

当$x\to\infty$时，亚里函数呈现出渐近线形式：

$$Ai(x) = \frac{1}{2\sqrt{\pi}}\frac{e^{-2/3x^{3/2}}}{x^{1/4}}$$

当$x\to-\infty$时，我们有：

$$Ai(x) = \frac{1}{\sqrt{\pi}}\frac{\sin\left(\frac{2}{3}x^{3/2}-\frac{\pi}{4}\right)}{x^{1/4}}$$

让我们使用**MATLAB**绘制亚里函数。首先我们选定能够显示出这个函数最主要特性的区间：

```
>> x = linspace(-10,5);
```

现在我们已经构建了一个可以被亚里函数使用的数组了：

```
>> y = airy(x);
```

绘制它：

```
>> plot(x,y),xlabel('x'),ylabel('Ai(x)'),grid on
Warning: Imaginary parts of complex X and/or Y arguments ignored.
```

结果如图11-13所示。可以看到，当x为负值时，函数有振荡行为，当x为正值时，函数呈现出指数衰减行为，这与刚才列出的渐近线公式一致。

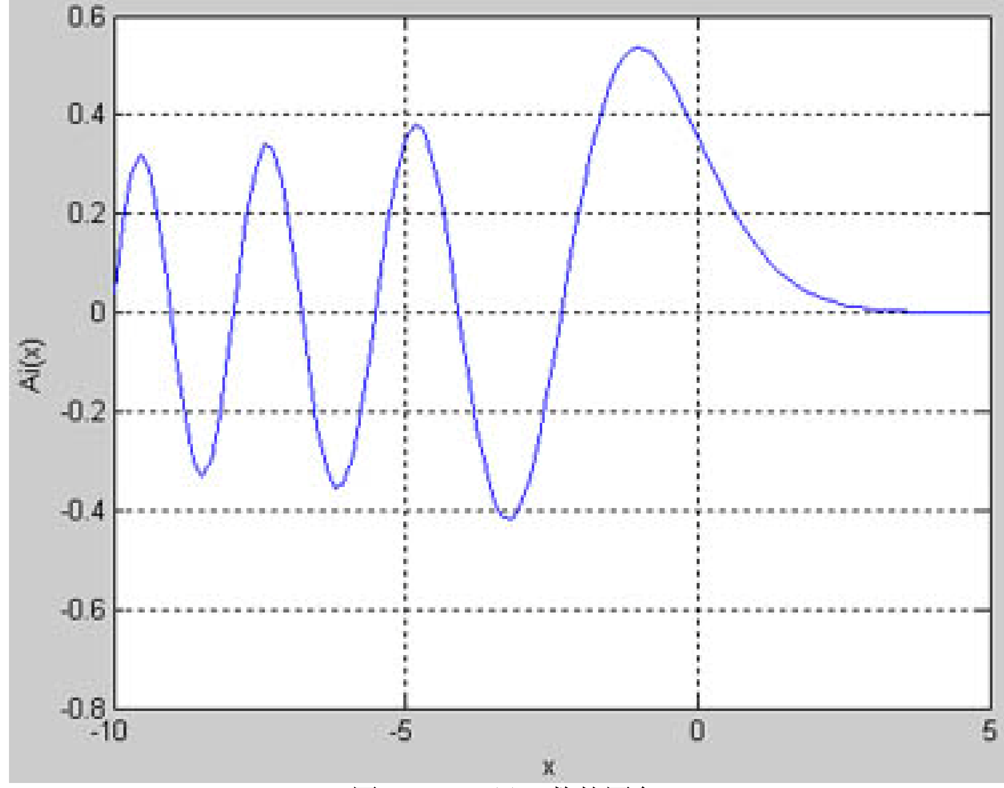


图 11-13 亚里函数的图象

习题

1.计算 18 伽马函数值。
2. 使用 **MATLAB** 中的伽马函数计算$\int_0^{\infty} 5^{-2z^2}dz$(使用替代技术用伽马函数重写积分)
3. 把 $1 \leqslant x \leqslant 2$ 区间内伽马函数值列成表格，x 的增量取 0.1。
4. 使用 **MATLAB** 求$\int_0^1 xJ_n(2x)J_n(x)dx$ 的值。
5. n维球体的体积公式是：

$$V_n = r^n \frac{\pi^{n/2}}{\Gamma(1 + n/2)}$$

编写一个**MATLAB**函数实现这个公式，并求出一个5维、半径为3m的球体体积。
6.计算下面的积分：

$$\int_0^{\pi/2} \sin^{13}\theta \cos^{15}\theta d\theta$$

7. 绘制相伴勒让德多项式$P_3^2(x)$的图象。
8. 计算积分：

$$\int_0^{\pi/2} \cos^{17}\theta d\theta$$

9. 创建一个表格，列出亚里函数在$0 \leqslant x \leqslant 5$内、增量取0.5时的值。
10. 检查亚里函数与伽马函数在$x = 0$时的关系。

【参考答案在第 243 页】



附录 A

最终测试

使用 **MATLAB** 计算下面的数值。

1. $4\frac{7}{8}-2^3+3(4^3)$。
2. $5(12^{3/4}-2)$ 。
3. $36^{0.78}$。
4. 使用 **MATLAB** 求：

$$\frac{d}{dx}(x^2\cos(\pi x))$$

5. 求$f(x)=\frac{2x}{x^2-9}$ 的一阶导数。
6. 求 $x\sin(x)$的二阶导数。
7. 使用 **MATLAB** 求$(1+x)^4$的二项展开式。
8. $f(x)=2x^3-3x^2$的临界点是什么？
9. 求极限： $\lim_{x\to 1}\frac{x^3-1}{x^2-1}$ 。
10. 求极限：$\lim_{x\to 0}3^{1/x}$。
11. 求极限：$\lim_{x\to 0^+}3^{1/x}$ 。
12. 求极限：$\lim_{x\to 0^-}3^{1/x}$ 。
13. 计算$\sqrt[4]{x}$ 的导数并求在 $x=67$ 时的值。
14. 把 $x=1.1$ 和 $x = 2.7$ 代入方程 $x^2-12x+4$。
15. $(\sqrt{x}+2)^{99}$ 的三阶导数是什么？
16. **MATLAB** 中的什么函数可以用来求一个方程的根？
17. 我们使用 **MATLAB** 求解方程 $2x + 3 = 0$。调用正确的函数和语法是：
    (a) `find('2 * x + 3')`
    (b) `roots('2 * x + 3')`



(c) `solve('2 * x + 3')`
(d) `solve('2 * x + 3', 0)`

18. 我们使用 **MATLAB** 求解方程 2*x* + 3 = 1。调用正确的函数和语法是：
(a) `roots('2 * x + 3 = 1')`
(b) `solve('2 * x + 3 = 1')`
(c) `solve('2 * x + 3', 1)`
(d) `roots('2 * x + 3', 1)`

19. 求 $x^2-5x+9=0$ 的根。

20. 使用 **MATLAB** 对 x^3-64 分解因式。

21. 计算极限 $\lim\limits_{x\to 2^+}\frac{x+3}{x-4}$ ，正确的 **MATLAB** 调用是：
(a) `limit((x + 3)/(x - 4), 2, +)`
(b) `limit((x + 3)/(x - 4)), 2,+`
(c) `limit((x + 3)/(x - 4), 2, 'right')`
(d) `limit((x + 3)/(x - 4), 2), right`

设函数 $f(x)=x^2$ 的定义域是[1, 3]，回答问题 22-24。

22. 最小值是什么？

23. 最大值是什么？

24. 求这个函数的均值。

25. 如果 **MATLAB** 输出函数 f 为：

```
f =
x^3 - 3*x^2 - 4*x + 2
```

我们能够使用哪个函数让它显示成 x^3-3x^2-4x+2？

26. 要产生 $0\leqslant x\leqslant 10$ 之间间隔均匀的数集，我们可以写成：
(a) `x = linspace(0:10);`
(b) `x = linspace(0:0.1:10);`
(c) `x = linspace(0, 10, 'u');`
(d) `x = linspace(0, 10);`

27. 如果你想要绘制两条关于 *x* 的曲线 *y*1 和 *y*2。能够把 *y1* 绘成红线而 *y2* 绘成蓝线的正确命令是：
(a) `plot(x, y1, 'r', x, y2, 'b')`
(b) `plot(x, y1, 'r', y2, 'b')`
(c) `plot(x, y1, "r", y2, "b")`
(d) `plot(x, y1, x, y2), color('r', 'b')`

28. 如果你要把某线曲线绘成蓝色虚线，那么正确的命令是：
(a) `plot(x, y, 'b-')`
(b) `plot(x, y, 'b-')`
(c) `plot(x, y, 'b', '-')`

29. 给图添加标题的正确命令是：
(a) `plot(x, y, 'title->Plot of Sin(x)')`
(b) `plot(x, y, 'Plot of Sin(x)')`
(c) `plot(x, y), title('Plot of Sin(x)')`
(d) `plot(x, y), Title('Plot of Sin(x)')`

30. 把某条曲线绘成红点线的正确命令是：
(a) `plot(x, y, 'r', ':')`
(b) `plot(x, y, 'r:')`
(c) `plot(x, y, 'r.')`

31. 要创建符号函数 $f(t) = \sin(t)$，最好使用的命令是：
(a) `syms t; f = sin(t);`
(b) `syms t; f = sin('t');`
(c) `syms t; f = 'sin(t)';`
(d) `a or c`

32. 要绘制符号函数 *f* 的图象，你调用的命令是：

(a) `quickplot( f )`
(b) `symplot( f )`
(c) `ezplot( f )`
(d) `plot( f, 'symbolic')`

33. 如果你想使用 *ezplot* 来产生函数 *f* 的图象并在**同一个图上**绘出它的二阶导数图象。正确的命令是：
(a) `ezplot( f, diff( f, 2))`
(b) `subplot(1,2,1); ezplot(f) subplot(1,2,2); ezplot(diff(f,2))`
(c) `ezplot( f ); hold on ezplot(diff( f, 2))`
(d) `ezplot( f ); hold; ezplot(diff( f, 2));`

34. 如果你想使用 *ezplot* 来产生函数 *f* 的图象，并且并排绘制它的二阶导数图象。正确的命令是：
(a) `subplot(1,2,1); ezplot(f) subplot(1,2,2); ezplot(diff(f,2))`
(b) `ezplot(subplot(1, f ), subplot(2, diff( f, 2))`
(c) `ezplot( f ); hold on ezplot(diff( f, 2))`

35. 假设你已经创建了一个符号函数 *f*。为了显示它的三阶导数，你可以写成：
(a) `f''''`
(b) `f''''` 或 `diff( f, 3)`
(c) `derivative( f, 3)`
(d) `diff( f, 3)`

36. *polyfit*(*x*, *y*, *n*)返回的是：
(a) 一个与用数组传递给 *polyfit* 的数集相拟合的符号多项式函数。
(b) 用于降低次数的、次数为 *n* 的拟合多项式系数。
(c) 用于提高次数的、次数为 *n* 的拟合多项式系数。
(d) 由数组指定的点上与多项式的拟合的数组。

37. *subplot* 命令：
(a) 允许你在同一个图上产生多个图象。
(b) 允许你在同一图上产生多条曲线。
(c) 在 **MATLAB** 并没有 *subplot* 命令。

38. 调用 *subplot*(*m*, *n*, *p*)
(a) 把图分成有 *m* 行 *n* 列的图象组，并把当前图象绘到窗格 *p* 上。
(b) 在窗格 *p* 中绘制 *m* 与 *n* 的关系曲线。
(c) 在(行，列) = (*m*, *n*)的位置绘制曲线 *p*。
(d) 在 **MATLAB** 并没有 *subplot* 命令。

39. 当编写 **MATLAB** 代码时，要在 *if* 语句中表明逻辑关系 *a* **不等于** *b*，你要写成：
(a) `a != b`
(b) `a.NE.b`
(c) `a <> b`
(d) `a ~= b`

40. 在 **MATLAB** 代码中**或(OR)操作符**用什么表示？
(a) `OR` 关键字
(b) 在变量间输入&
(c) 在变量间输入~
(d) “导管”字符|

41. 如果 ***A*** 是一个列向量，要对它进行转置创建行向量 ***B***，你要写成：
(a) `B = transpose(A)`
(b) `B = A'`
(c) `B = t(A)`
(d) `B = trans(A)`

42. 设 *x* = 7 和 *y* = -3。**MATLAB** 中的语句 *x* ~= *y* 将产生：
(a) `ans = 1`
(b) `ans = 0`

(c) ans = -1
(d) 错误

43. 要把元素 1 -2 3 7 输进一个列向量中，我们输入：
(a) [1: -2: 3: 7]
(b) [1, -2, 3, 7]
(c) [1; -2; 3; 7]
(d) [1 -2 3 7]

44. 要计算数据向量 *f* 的平方根，写成：
(a) sqr(f)
(b) f. ^ 2
(c) f ^ 2
(d) square(f)

45. 为了得到函数 *f* 在原点处的第四次项泰勒展开式，我们使用：
(a) Taylor(f, 4)
(b) taylor(f, 4)
(c) taylor(f), terms('4')
(d) taylor, f, 4

46. 用来在屏幕上显示文本字串 s 的命令是：
(a) disp(s)
(b) display(s)
(c) display, s
(d) Disp(s)

47. 函数 *f = a*x^2 - 2*，则命令 *solve*(*f, a*)产生的结果是：
(a) 2/a
(b) 2/x
(c) -2/x
(d) 错误：rhs must be specified.

48. **MATLAB** 中的 *ode23* 和 *ode45* 解算器：
(a) 都基于 *Runge-Kutta*(龙格-库塔)方法
(b) 都基于 *Euler*(欧拉)方法
(c) 使用拉格朗日乘子法(*method of Lagrange multipliers*)
(d) 都基于弛豫(*relaxation based*)

49. 一个时间上的函数 *y = y*(*t*)是某个微分方程的解。使用 *ode23* 求解方程得到是：
(a) 一个具有[*t y*]形式的数组。
(b) 符号解 *y = y*(*t*)。
(c) 错误，**MATLAB** 中没有 *ode23* 解算器。
(d) 一个具有[*y t*]形式的数组。

50. 下面命令可以求得向量 ***v*** 中所有元素的均值的是：
(a) Ave(v)
(b) ave(v)
(c) mean(v)
(d) average(v)

51. 命令 *pinv*
(a) 是一个矩阵的特征逆
(b) 产生一个矩阵的伪逆
(c) 只能使用符号数据来调用

52. 使用 **MATLAB** 求：

$$\frac{d^{10}}{dt^{10}}(e^{-0.1t}\cos 2t)$$

53. 变量 y 有 1 行 11 列。如果我们输入 size(y)，**MATLAB** 返回：
(a) 11
(b) 1 11

(c) `11 1`
(d) `(1, 11)`

54. 要实现 1≤x≤5 内步长为 0.5 的 for 循环，最好的命令：
(a) `for i = 1, 5, 0.5`
(b) `for i = 1:0.5:5`
(c) `for loop i = 1, 5, 0.5`
(d) `loop i = 1:0.5:5 do`

55. 函数 $y=e^{-2x}$ 是定义在某数值数据 *x* 上的，要输入它，使用的是：
(a) `y = exp(-x)*exp(-x)`
(b) `y = sqr(exp(-x))`
(c) `y = exp(-x).*exp(-x)`
(d) `y = sqrt(exp(-x))`

56. 为了求得某个列向量 x 的最大值，输入：
(a) `max(x)`
(b) `maximum(x)`
(c) `largest(x)`

57. **MATLAB** 中用来在 *x*-*y* 图象上添加标签的是：
(a) `labels` 命令。
(b) `xlabel` 和 `ylabel` 命令。
(c) `label` 命令。
(d) `text` 命令。

58. 假设我们执行了 *y = polyval(n, x)*，那么 *polyval* 返回：
(a) 没有 *polyval* 命令。
(b) 一个在点 *x* 上的 *n* 次多项式。
(c) *n* 次多项式系数。

59. 用于指定 *x*-*y* 图象中的定义域 $a \le x \le b$ 和值域 $c \le y \le d$ 的是：
(a) `axis([a b c d])`
(b) `axis(a b c d)`
(c) `axis([a, b, c, d])`
(d) `axis a, b, c, d`

60. 绘制对数图象的正确命令是：
(a) `plot(x, y, 'log-log')`
(b) `loglog(x, y)`
(c) `log(x, y)`
(d) `logplot(x, y)`

使用 **MATLAB** 计算问题 61-70 中的积分：

61. $\int \frac{dx}{x\sqrt{1+x^2}}$

62. $\int (\sin x^2) 2x dx$

63. $\int e^{x^5} x^4 dx$

64. $\int \frac{\cos(\ln(x))}{x} dx$

65. $\int \ln x^2 dx$

66. $\int_0^2 x^3 dx$

67. $\int_0^\infty x e^{-2x} dx$

68. $\int_{-\infty}^\infty x e^{-2x} dx$

69. $\int_0^b \sinh(x) dx$

70. $\int_{2\pi}^{3\pi} \sin t \sin^2 4t dt$

71. 求 $y=x^2$ 和 $y=e^{-x}$ 这两条曲线在 $0\leqslant x\leqslant 1$ 上所包含的面积
72. 求 $y=x^2$ 和 $y=-3/2x$ 在 $0\leqslant x\leqslant 2$ 上之间的面积。
73. 使用 **MATLAB** 计算

$$\int_0^1\int_{-1}^1\int_0^2 x^2e^{-2xy}z\,dxdydz$$

74. 使用 **MATLAB** 求 $f(r,\theta,\varphi)=r\sin 2\theta\cos\varphi$ 在半径为 2 的球体内的积分。
75. 当使用 **MATLAB** 求解微分方程时，*dsolve* 命令要求用户指明函数 $y=f(x)$的二阶导数，输入为：
    (a) `y''`
    (b) `diff(y, 2)`
    (c) `D^2y`
    (d) `D2y`
    (e) `D(Dy)`
76. 为了得到下面方程的解

$$\frac{dy}{dt}+2y=0$$

在 **MATLAB** 中最好使用的语法是：
    (a) `diff('y' + 2*y = 0')`
    (b) `dsolve('Dy + 2*y')`
    (c) `dsolve('Dy + 2*y = 0')`
    (d) `diff(Dy + 2y)`

用 **MATLAB** 求出问题 77-81 中给定的微分方程的符号解。

77. $\frac{dy}{dt}+2y=0$
78. $\frac{dy}{dt}+2y=-1$
79. $\frac{d^2y}{dx^2}+4y-2=0$
80. $\frac{d^3y}{dx^3}+y=0$
81. $\frac{d^2f}{dt^2}+2f=4\cos 3t$

求问题 82-90 中符合 *IVP* 和 *BVP* 条件的解，然后把绘制成图象。

82. $\frac{dy}{dt}-3y=0, y(0)=1.$
83. $\frac{dy}{dt}-3y=2, y(0)=-1.$
84. $\frac{dy}{dt}+y=\cos t, y(0)=2.$
85. $\frac{d^2y}{dt^2}-3y+1=0, y(0)=0, y'(0)=1.$
86. $\frac{d^2y}{dt^2}+y=0, y(1)=0, y'(2)=-1.$
87. $\frac{d^2y}{dt^2}+y+4=0, y(0)=1, y'(0)=1.$
88. $\frac{d^2y}{dt^2}+y+4=A, y(0)=1, y'(0)=1.$
89. $\frac{d^2y}{dt^2}+y+4=2e^{-t}, y(0)=1, y'(0)=0.$
90. $\frac{d^2y}{dt^2}+y+4=\sin(5t), y(0)=0, y'(0)=1.$
91. 使用 **MATLAB** 求解方程组：

$$\frac{dx}{dt}=x-2y$$
$$\frac{dy}{dt}=2x+y$$

92. 取 x(0)=0,y(0)=1，求方程组的解。
93. 求矩阵的逆

$$\mathbf{A}=\begin{pmatrix}1 & 2\\ 2 & 1\end{pmatrix}$$

94. 通过计算系数矩阵的逆矩阵解方程组：

$$x+2y=4$$
$$3x-4y=7$$

95. 求矩阵的秩：

$$\mathbf{B}=\begin{pmatrix}1 & -2 & 4\\ 3 & 2 & -5\\ 1 & 2 & 8\end{pmatrix}$$

96. 求前一问题中矩阵的逆。
97. 求问题 95 中矩阵 B 的 LU 分解。
98. 求问题 95 中矩阵 B 的本征值。
99. 求函数 f 在 $0\leqslant t\leqslant 5$ 范围内的数值积分，其中 f 有数据点 f = [1, 2, 5, 11, 8]。
100. 对在 $0\leqslant x\leqslant 1$ 上的 $\cos(x^2)$ 进行数值积分。

附录 B



习题及测试答案

第一章：MATLAB 环境

1. 3.9286
   ```
   >> 5*11/14
   ```
2. 2.2×10^3
   ```
   >> 5*8/3+3^7
   ```
   或
   ```
   >> 5*8/3+power(3,7)
   ```
3. 15.5885
   ```
   >> 9^1.25
   ```
   或
   ```
   >> power(9,1.25)
   ```
4. 假
5. 150.7964 立方厘米
   ```
   >>r=2;h=12;v=pi*r^2*h
   ```
6. 1170/1351
   ```
   >> format rat; sin(pi/3)
   ```
7.
   ```
   % File to calculate sin of numbers
   x = [pi/4,pi/3,pi/2];
   format rat
   y = sin(x)
   ```

第二章：向量与矩阵

1. 7.94
   ```
   >> a=[-1,7,3,2];sqrt(sum(a.*a))
   ```
   或
   ```
   >> a=[-1,7,3,2];sqrt(dot(a,a))
   ```
2.
   ```
   >> A = [-1+i 7*i 3 -2-2*i];
   >> sqrt(dot(A,A))
   ans =
      8.2462
   ```
3.
   A = [1; 2; 3]; B = [1 2 3] 或者 B = [1, 2, 3]
4. [4; 10 ; 18]
   ```
   >> A=[1;2;3]; B=[4;5;6];
   >> A.*B
   ```
5. eye(5)
6. 数组相乘＝$\begin{pmatrix} 16 & 7 & 22 \\ -6 & 30 & -4 \\ 0 & 4 & -16 \end{pmatrix}$，矩阵相乘＝$\begin{pmatrix} 31 & 72 & 66 \\ 5 & 34 & 30 \\ -18 & -4 & -8 \end{pmatrix}$
   ```
   >> A=[8,7,11;6,5,-1;0,2,-8]
   >> B=[2,1,2;-1,6,4;2,2,2]
   >> A.*B
   >> A*B
   ```
7. 输入下面的命令：
   ```
   >> A = [1 2 3; 4 5 6; 7 8 9]
   A =
   ```



```
     1     2     3
     4     5     6
     7     8     9
>> B = A([3,3,2],:)
B =
     7     8     9
     7     8     9
     4     5     6
```

8.

```
>> D=[1,2,3;-4,1,2;0,9,-8];
>> b=[12;13;-1];
>> D\b
ans =
   -1.2424
    2.3939
    2.8182
>> det(D)
ans =
  -198
```

9.

```
>> A=[1,-2,3;1,4,3;2,8,1];
>> b=[1;2;3];
>> det(A)
ans =
   -30
>> A\b
ans =
    0.7333
    0.1667
    0.2000
```

或

```
>> inv(A)*b
ans =
    0.7333
    0.1667
    0.2000
```

10.

```
>> A=[1,7,-9;2,-1,4;1,1,-7];
>> b=[12;16;16];
>> [L,U]=lu(A)
L =
    0.5000    1.0000         0
    1.0000         0         0
    0.5000    0.2000    1.0000
U =
    2.0000   -1.0000    4.0000
         0    7.5000  -11.0000
         0         0   -6.8000
>> x=U\(L\b)
x =
    9.6078
   -1.0196
   -1.0588
```

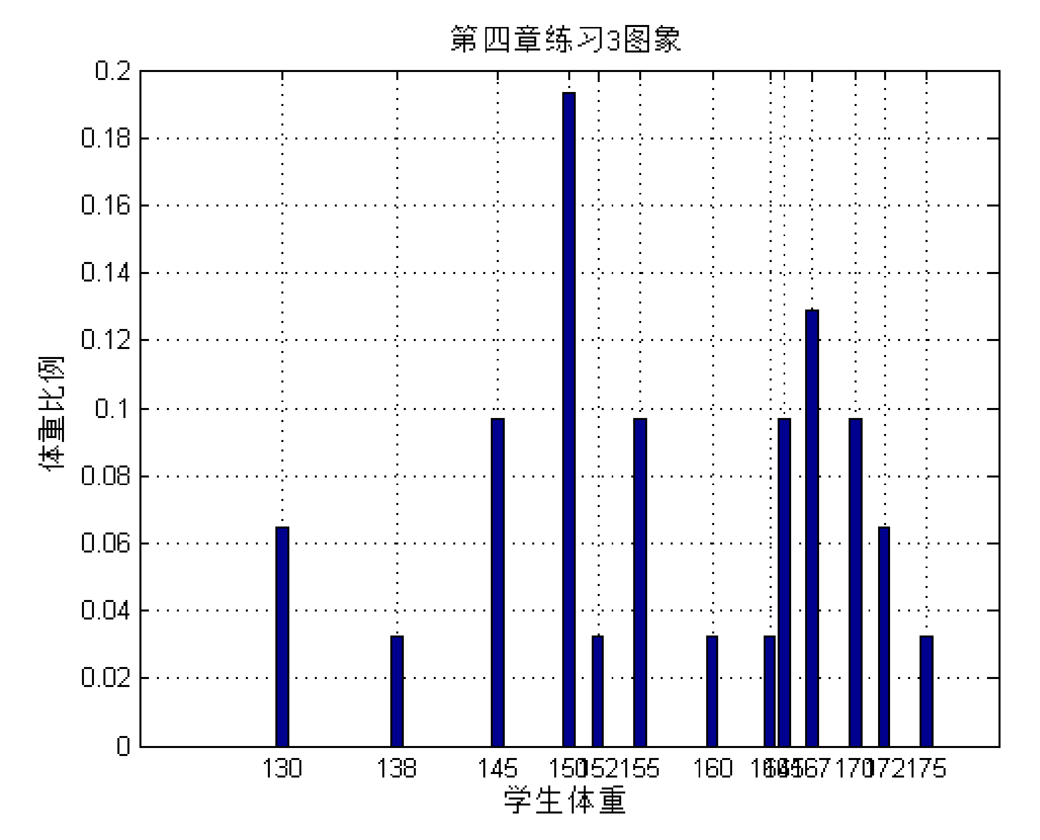
第三章：绘图与图形

1.
```
>> x = [0:0.1:1];
>> y = tan(x);
>> plot(x,y),xlabel('x'),ylabel('tan(x)')
```
2.
```
>> z = sin(x);
>> plot(x,y,x,z),xlabel('x'),ylabel('y')
```
3.
```
x = [-pi: 0.2: pi], x = linspace(-pi, pi), x = linspace(-pi, pi, 50)
```
4.
```
[x, y] = meshgrid(-3:0.1:2, -5:0.1:5)
[x, y] = meshgrid(-5:0.2:5)
```
5.
```
>> t = [0: pi/40:10*pi];
>> plot3(exp(-t).*cos(t), exp(-t).*sin(t),t), grid on
```

第四章：统计和 MATLAB 编程介绍

1. 均值 = 156.9677 磅， 中位数 = 155 磅， 标准差=12.2651 磅。
```
>> x=[130,138,145,150,152,155,160,164,165,167,170,172,175];
>> y=[2,1,3,6,1,3,1,1,3,4,3,2,1];
>> ave=mean(sum(x.*y)/sum(y))
>> raw = [];
for i = 1:length(x)
  if x(i) > 0
    new = x(i)*ones(1,y(i));
  else
    new = [];
  end
  raw =[raw,new];
end
>> median(raw)
>> std(raw)
```
2. 学生重 150 磅的概率是 0.1935。
```
>> y(4)/sum(y)
```
3. 大约 0.07。
```
>> bar(x,y/sum(y));grid on;xlabel('学生体重');
>> ylabel('体重比例');title('第四章练习 3 图象')
```
4. 代码是：
```
function cylinder_volume
%要求用户输入半径
r = input('请输入半径：');
%获取高度
h = input('请输入高度：');
vol = pi*r^2*h;
disp('体积是：');
disp(vol)
```





5.

```
x = input('输入 x: ');
n = input('输入最高指数 n: ');
i = 0;
sum = 0;
while i <= n
    sum = sum + x^i;
    i = i+1;
end
```

6.

```
n = input('输入指数 n: ');
sum=0;
for r = 0:n
    sum = sum + 1/x^r;
end
```

第五章：代数方程求解和其它符号工具

1. -14.6882

```
>> d='x=7*sqrt(2)-5*sqrt(60)+5*sqrt(8)'
>> double(solve(d))
ans =
      -14.688
```

2. $\frac{-1\pm\sqrt{22}}{3}$

```
>> solve('3*x^2+2*x=7')
ans =
 - 22^(1/2)/3 - 1/3
```

```
  22^(1/2)/3 - 1/3
```

3. $\frac{-1\pm\sqrt{5+4\pi}}{2}$

```
>> solve('x^2+sqrt(5)*x-pi')
ans =
 - 5^(1/2)/2 - (4*pi + 5)^(1/2)/2
   (4*pi + 5)^(1/2)/2 - 5^(1/2)/2
```

4. x = 5/2, ezplot('sqrt(2*x - 4) - 1', [2, 4, 0, 1])。

```
>> x=solve('sqrt(2*x-4)=1')
```

5. 根是 1.1162, -0.3081 + 1.6102i 和-0.3081 - 1.6102i。

```
>> solve('2*t^3-t^2+4*t-6','t');
>> double(ans)
ans =
   1.1162
  -0.3081 - 1.6102i
  -0.3081 + 1.6102i
```

6. x = 1, y = -3, z = 2。

```
>> eq1='x-3*y-2*z=6';
>> eq2='2*x-4*y-3*z=8';
>> eq3='-3*x+6*y+8*z=-5';
>> s=solve(eq1,eq2,eq3);
>> s.x
ans =
1
>> s.y
ans =
-3
>> s.z
ans =
2
```

7. 一个实根，x = -0.7035。

```
>> double(solve('exp(x)-x^2'))
ans =
   1.5880 - 1.5402i
  -0.7035
```

8. -1。

```
>> syms x
>> simplify(tan(x)^2-sec(x)^2)
ans =
-1
```

9. x + 1/3*x^3 + 2/15*x^5 + 17/315*x^7 + 62/2835*x^9。

```
>> syms x
>> taylor(tan(x),10)
ans =
x+1/3*x^3+2/15*x^5+17/315*x^7+62/2835*x^9
```

10. 1 - 1/8*x^2 + 5/192*x^4 - 113/23040*x^6 + 241/258048*x^8。

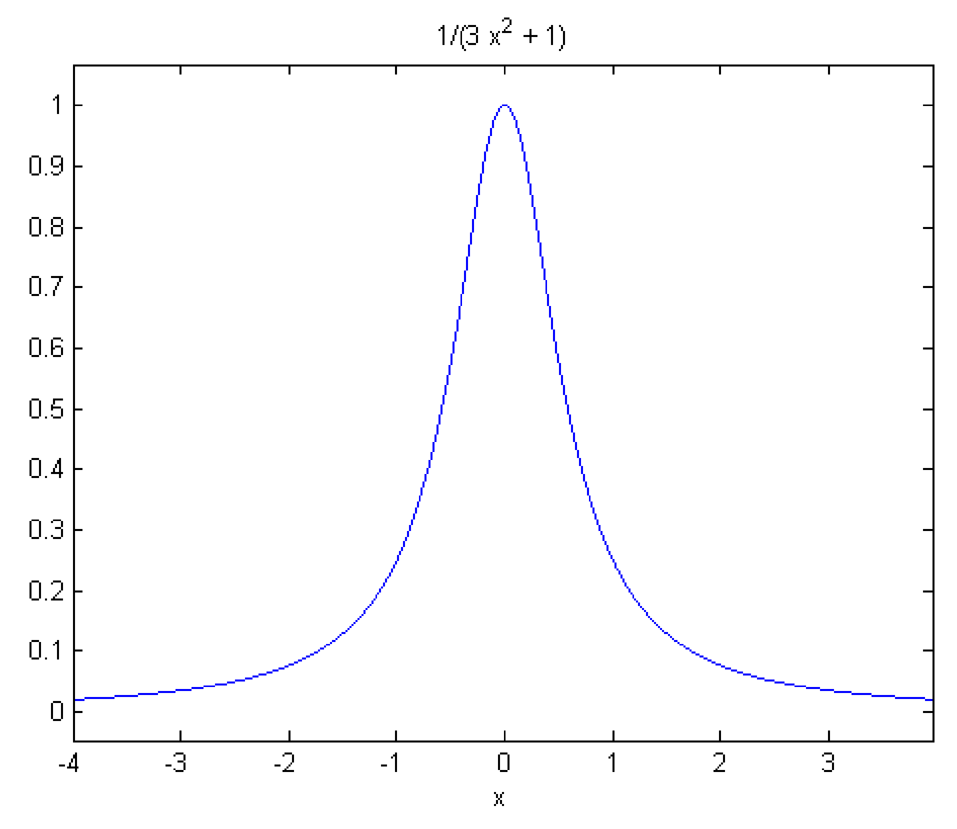
```
>> syms x
>> taylor(4/(5-cos(x)),10)
ans =
1-1/8*x^2+5/192*x^4-113/23040*x^6+241/258048*x^8
>> ezplot(ans)
```



第六章：基本符号演算和微分方程

1. 4
```
>> syms x
>> limit(sin(4*x)/x,0)
ans =
4
```
2. 极限不存在。计算得到的左极限和右极限分别是-1 和+1。
```
>> syms x
>> f=(x-2)/abs(x-2);
>> limit(f,x,2,'left')
ans =
-1
>> limit(f,x,2,'right')
ans =
1
```
3. 渐近线在 0 和 3。
```
>> syms x
>> f=x/(x^2-3*x);
>> ezplot(f);
>> s=solve(x^2-3*x)
s =
 0
 3
>> hold on
>> plot(double(s(1))*[1 1],[2,-2],'--r')
>> plot(double(s(2))*[1 1],[2,-2],'--r')
>> hold off
%这里 x＝0 是第一类可去间断点
```
4. 0
```
>> syms theta
>> f=cos(theta)/(1+sin(theta));
>> limit(f,theta,pi/2)
ans =
0
```
5.

(a) -(6*x)/(3*x^2 + 1)^2, (72*x^2)/(3*x^2 + 1)^3 - 6/(3*x^2 + 1)^2
```
>> syms x;
>> f=1/(3*x^2+1);
>> g=diff(f)
g =
-(6*x)/(3*x^2 + 1)^2
>> h=diff(g)
h =
(72*x^2)/(3*x^2 + 1)^3 - 6/(3*x^2 + 1)^2
```
(b) x = 0
```
>> x=solve(g)
x =
0
```
(c) f''(0) = -6
```
>> limit(h,x)
ans =
-6
```
(d) 由于 f"(0) < 0,所以 x = 0 局部最大值。
```
>> ezplot(f)
```



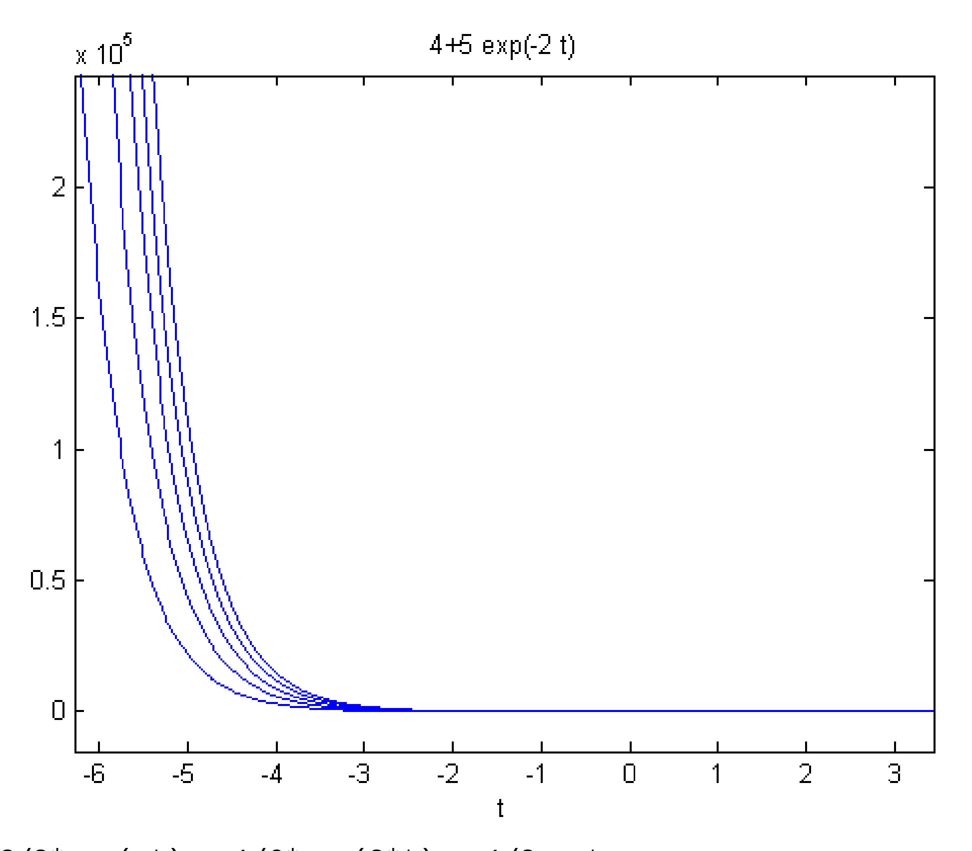
6. diff(y, 2) - 11*y = -24*sin(t) + 36*cos(t) ≠ -4cos6t

```
>> syms t;
>> y=2*sin(t)-3*cos(t);
>> f=-4*cos(6*t);
>> a=diff(y,2)-11*y;
>> isequal(a,f)
ans =
     0
```

7. 4 + exp(-2*t)*C1

```
>> f='Dx=-2*x+8';
>> s=dsolve(f)
s =
4+exp(-2*t)*C1
>> for i=1:5
f=subs(s,'C1',i);
ezplot(f);
hold on;
end
```





8. -2/3*exp(-t) + 1/6*exp(2*t) + 1/2 – t

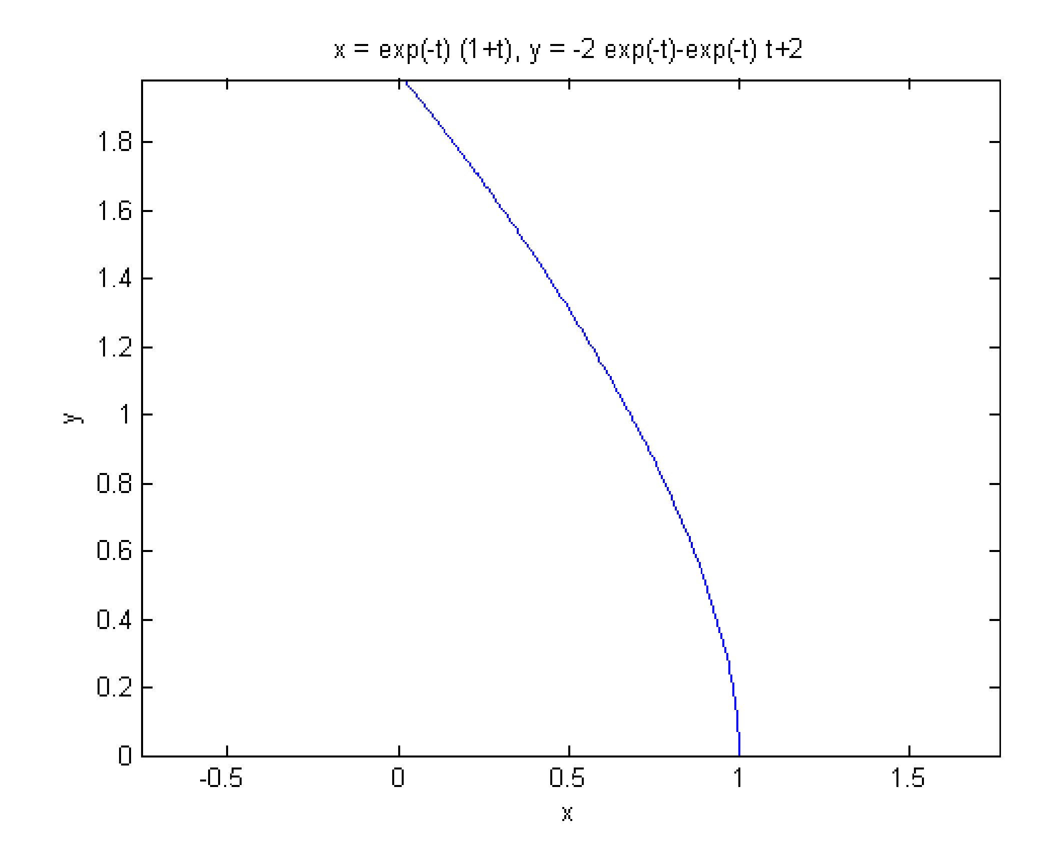
```
>> f='D2y-Dy-2*y=2*t';
>> s=dsolve(f,'Dy(0)=0','y(0)=0')
s =
-2/3*exp(-t)+1/6*exp(2*t)+1/2-t
```

9. x = 4*cos(t), p = -2*cos(t) - 2*sin(t) + 2*exp(-t)。

```
>> f1='D2x+x=0';
>> f2='Dp+p+x=0';
>> s=dsolve(f1,f2,'x(0)=4','Dx(0)=0','p(0)=0')
s =
    p: [1x1 sym]
    x: [1x1 sym]
>> s.x
ans =
4*cos(t)
>> s.p
ans =
-2*cos(t)-2*sin(t)+2*exp(-t)
```

10. x = exp(-t)*(1 + t), p = -2*exp(-t) - exp(-t)*t + 2

```
>> f1='D2x+2*Dx+x=0';
>> f2='Dp=x';
>> s=dsolve(f1,f2,'x(0)=1','Dx(0)=0','p(0)=0')
s =
    p: [1x1 sym]
    x: [1x1 sym]
>> x=s.x
x =
exp(-t)*(1+t)
>> p=s.p
p =
-2*exp(-t)-exp(-t)*t+2
>> ezplot(s.x,s.p)
```



第七章：ODE 的数值解

1. 0

```
function xdot=C7Ex1(t,x);
%用于第 7 章练习 1 的函数
xdot=zeros(2,1);
xdot(1)=2*x(1)*x(2);
xdot(2)=-x(1)^2;
%函数结束
>> [t,y]=ode23('C7Ex1',[0 1],[0 0]);
>> plot(t,y,'o');
```

2. $x_1 = \cos t + \sin t$, $x_2 = -\sin t + \cos t$

```
function xdot=C7Ex2(t,x);
%用于第 7 章练习 2 的函数
xdot=zeros(2,1);
xdot(1)=x(2);
xdot(2)=-x(1);
%函数结束
>> [t y]=ode23('C7Ex2',[0 1],[1 1]);
>> plot(t,y); grid on;
```

3. 解的图象如图 B7-1 所示。

```
function ydot=C7Ex3(t,y);
%用于第 7 章练习 3 的函数
ydot=-2.3*y;
%函数结束
>> [t,y]=ode23('C7Ex3',[0 1],2);
>> plot(t,y);
```



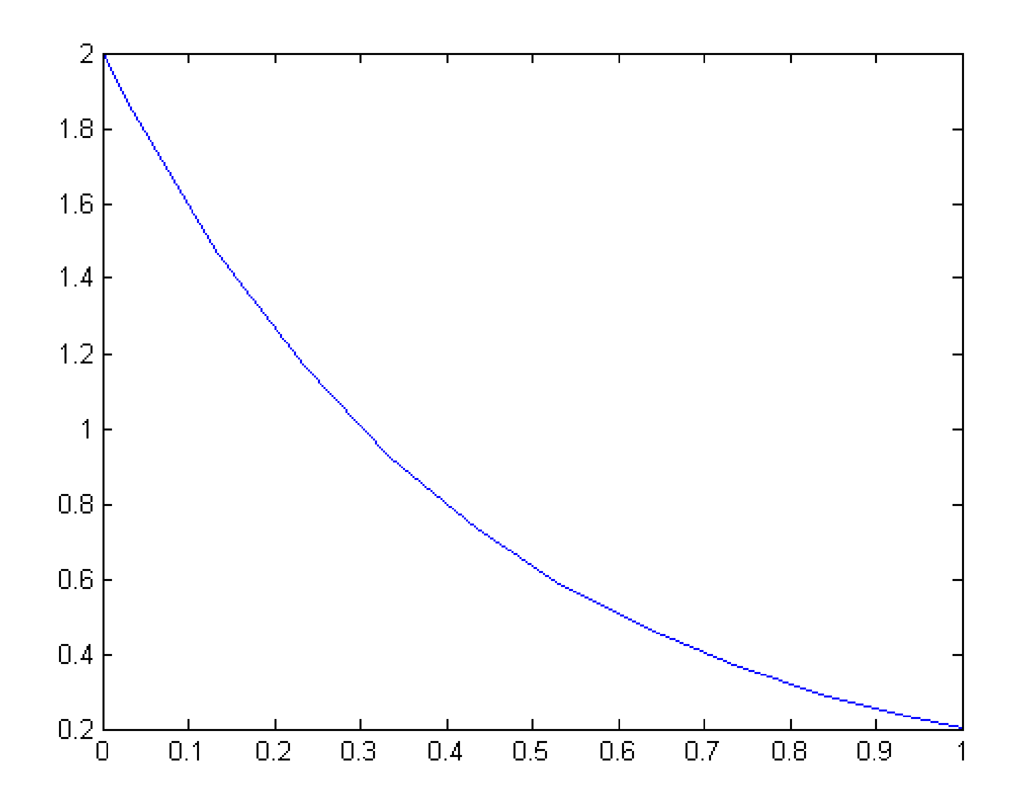


图 B7-1 第七章习题 3 方程组的解

4. 12 点和 41 点。

```
function ydot=C7Ex4(t,y);
%用于第 7 章练习 4 的函数
ydot=y;
%函数结束
>> [t23,y23]=ode23('C7Ex4',[0 1],1);
>> length(y23)
ans =
    12
>> [t45,y45]=ode45('C7Ex4',[0 1],1);
>> length(y45)
ans =
    41
>> plot(t45,y45);
```

5. 解已经绘在图 B7-2 中。

```
function ydot=C7Ex5(t,y);
%用于第 7 章练习 5 的函数
ydot=-t*y+1;
%函数结束
>> [t,y]=ode23('C7Ex5',[0 1],1);
>> plot(t,y);
```

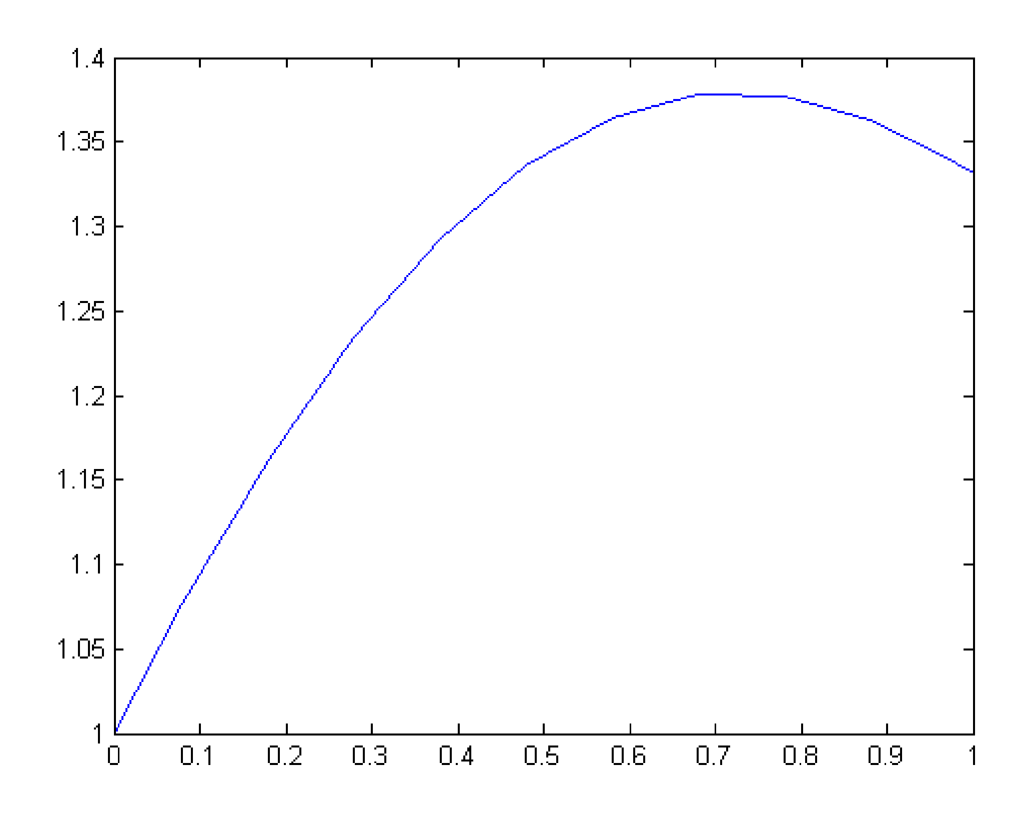


图 B7-2 第七章习题 5 的解

6. 解已经显示在图 B7-3 中。

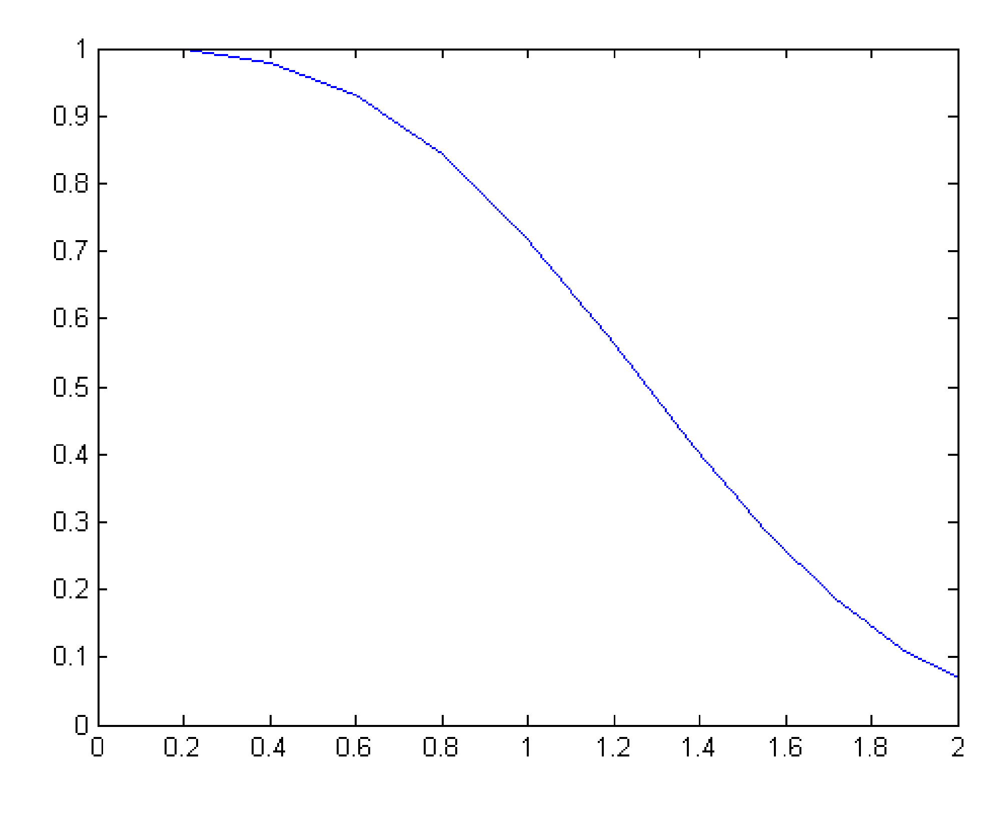


图 B7-3 第七章习题 6 的解

```
function ydot=C7Ex6(t,y);
```



```
%用于第 7 章练习 6 的函数
ydot=-t^2*y;
%函数结束
>> [t,y]=ode23('C7Ex6',[0 2],1);
>> plot(t,y);
```

7. 龙格-库塔(Runge-Kutta)
8. 解如图 B7-4 所示。

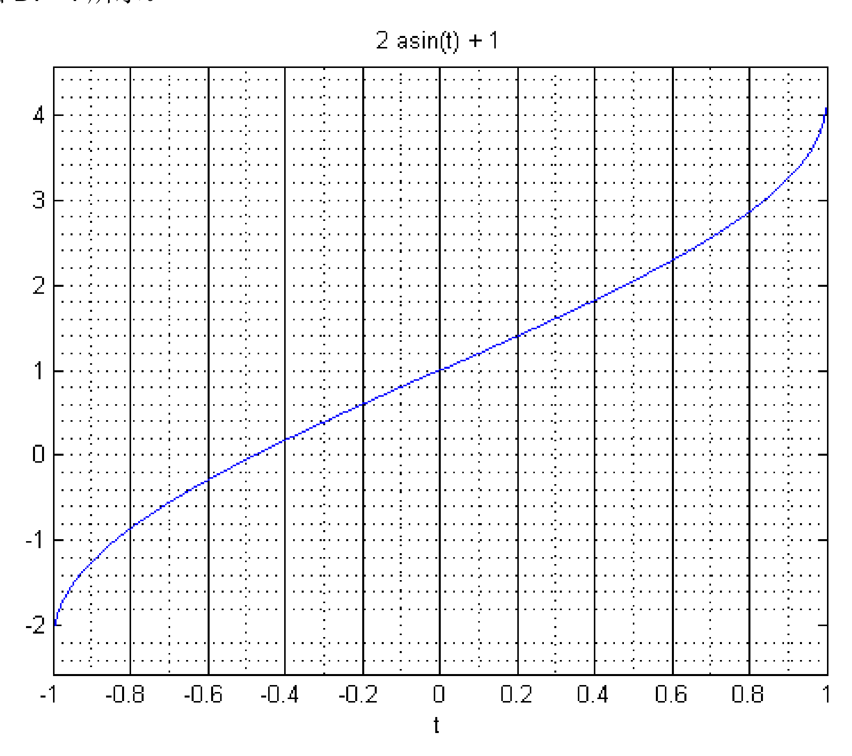


图 B7-4 $\frac{dy}{dt}=\frac{2}{\sqrt{1-t^2}}$, -1<t<1, y(0)=1解的图象

```
%本题不能像下面那样题求解：
function ydot=C7Ex8(t,y);
ydot=2/sqrt(1-t^2);
%然后在命令窗口中输入
>> [t y]=ode23('C7Ex8',[-1+eps 1-eps],1);plot(t,y);
%这样虽然能得到一个图象，但却是错误的。可以看出：y(0) ≠1，这与所给初始
%条件矛盾。原因在于 ode23 的第三个参数是所给区间起始点的值，
%即表示 y(-1+eps)=1。
%因此这里采用符号求解法求解析解：如下
>> s=dsolve('Dy=2/sqrt(1-t^2)','y(0)=1')
s =
    2*asin(t)+1
>> ezplot(s,[-1 1]);grid minor;
%经验证，结果正确。另本书原英文版所给答案有误，感兴趣的读者可下载英文版对照。
```

9. 解显示在图 B7-5 中。

```
function xdot=C7Ex9(t,x);
%用于第 7 章练习 9 的函数
%设 x1=y,x2=y',则 x1'=x2,x2'=y''=exp(-t)-x1+2x2;
xdot=zeros(2,1);
xdot(1)=x(2);
xdot(2)=exp(-t)-x(1)+2*x(2);
%函数结束
>> [t,y]=ode23('C7Ex9',[0 10],[2 0]);
```

```
>> plot(t,y);
```

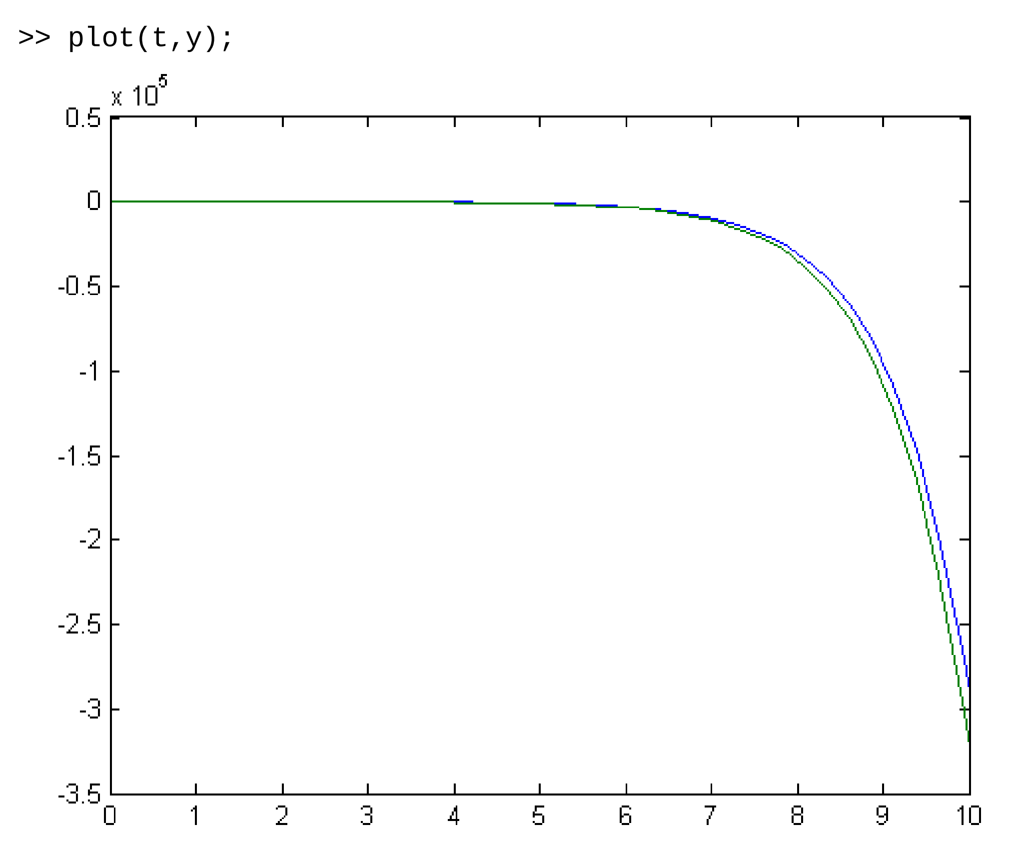


图 B7-5 $y'' - 2y' + y = \exp(-t), y(0) = 2, y'(0) = 0$ 的解

10. 图象显示在图 B7-6 中。

```
>> plot(y(:,1),y(:,2));
```

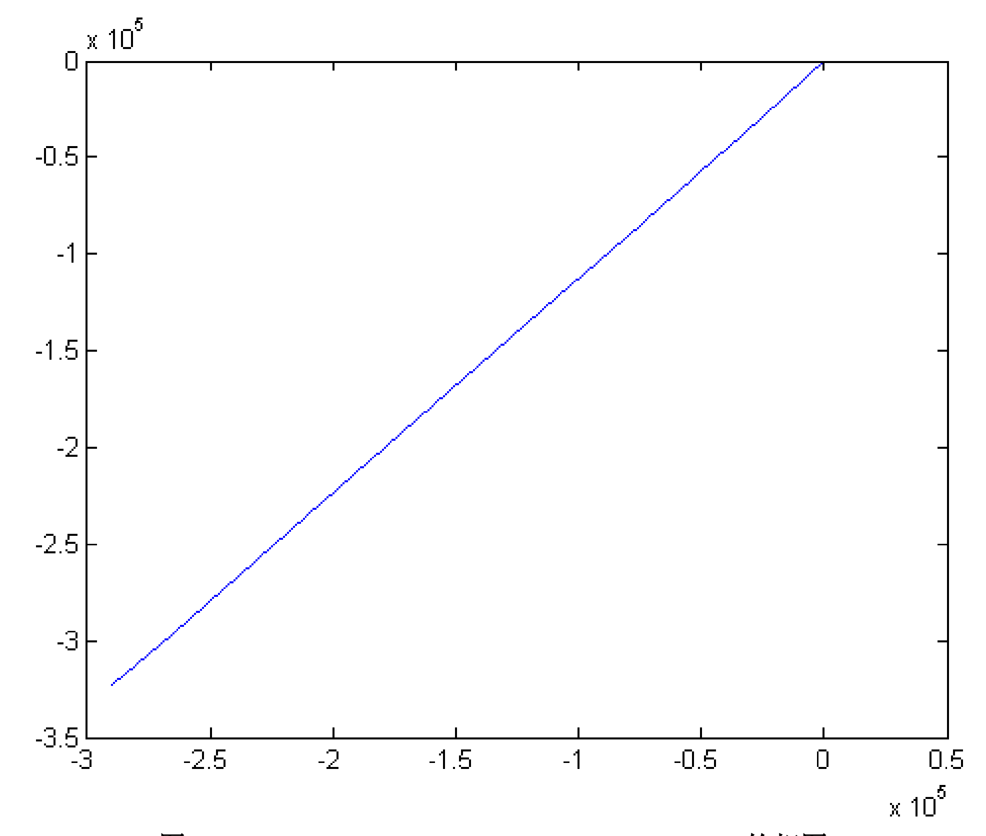


图 B7-6 $y'' - 2y' + y = \exp(-t), y(0) = 2, y'(0) = 0$ 的相图

第八章：积分

1.
```
1/a^2 * (-a*x*exp(-a*x) - exp(-a*x))
>> syms x a;
>> f='x*exp(-a*x)';
>> int(f)
ans =
1/a^2*(-a*x*exp(-a*x)-exp(-a*x))
```
2.
```
log(1/2*b + x + (x^2 + b*x + c)^(1/2))
>> syms x b c;
>> f='1/sqrt(x^2+b*x+c)';
>> int(f)
ans =
log(1/2*b+x+(x^2+b*x+c)^(1/2))
```
3.
```
1/2*i*x^2 - x*log(1 + exp(2*i*x)) + 1/2*i*polylog(2, -exp(2*i*x))
>> syms x;
>> f='x*tan(x)';
>> int(f)
ans =
1/2*i*x^2-x*log(1+exp(2*i*x))+1/2*i*polylog(2,-exp(2*i*x))
```
4.
```
1/2*y - 1/4*sin(2*x*y)/x
>> syms x y;
>> f='sin(y*x)^2';
>> int(f,'y')
ans =
1/x*(-1/2*cos(x*y)*sin(x*y)+1/2*x*y)
(经验证，matlab 输出的这个结果与原书所给答案是一致的，如下展开验证
>> syms x y;
>> expand(1/x*(-1/2*cos(x*y)*sin(x*y)+1/2*x*y))
ans =
1/2*y-1/2/x*cos(x*y)*sin(x*y)
>> expand(1/2*y - 1/4*sin(2*x*y)/x)
ans =
1/2*y-1/2/x*cos(x*y)*sin(x*y)
)
```
5.
```
3/10
>> syms x;
>> f='exp(-3*x)*cos(x)';
>> int(f,0,inf)
ans =
3/10
```
6.
```
0.9425
>> result=int('pi*x-pi*x^4',0,1)
result =
3/10*pi
>> double(result)
ans =
    0.9425
```
7.
```
>> syms theta r a phi;
>> V=int(int(int(r^2*sin(theta),r,0,a),theta,0,pi),phi,0,2*pi)
V =
4/3*a^3*pi
>> subs(V, a, 2)
ans =
   33.5103 立方米.
```

8. trapz = -0.1365，相对误差 0.34%.

```
>> x=linspace(0,pi,50);
>> y=cos(pi.*x);
>> in=trapz(x,y)
in =
   -0.1365
>> ac=double(int('cos(pi*x)',0,pi))
ac =
   -0.1370
>> err=abs((ac-in)/ac*100)
err =
    0.3383
```

9. 相对误差是 0.18629097599734，实质上不改变。

```
>> format long
>> x=linspace(-2.2,2.2,1000);
>> y=exp(-x.^2);
>> in=trapz(x,y)
in =
   1.76915192932756
>> ac=int('exp(-x^2)',-inf,inf)
ac =
pi^(1/2)
>> err=double(abs((ac-in)/ac*100))
err =
   0.18629097599736
```

10. 求积分得到 1.2459.

```
>> x=linspace(0,10,1000);
>> y = besselj(1,x);
>> trapz(x,y)
ans =
    1.2459
```


第九章：变换

1. 1/(s + 3)^2

```
>> syms t
>> g=t*exp(-3*t);
>> laplace(g)
ans =
    1/(s+3)^2
```

2. 40/(s^2 + 25) - 1/4*(s + 1)/(1/4*(s + 1)^2 + 1)

```
>> syms t;
>> f=8*sin(5*t)-exp(-t)*cos(2*t);
>> laplace(f)
ans =
    40/(s^2+25)-1/4*(s+1)/(1/4*(s+1)^2+1)
```

3. 1 - 2*dirac(t) + 5*sin(3*t) (1 是海维赛德函数 Heaviside function)

```
>> syms s;
>> F=1/s-(2*s^2+3)/(s^2+9);
>> ilaplace(F)
ans =
    1-2*dirac(t)+5*sin(3*t)
```

4. cosh(7*t) - sinh(3*t)

```
>> syms s;
```



```
>> F=s/(s^2-49)-3/(s^2-9);
>> ilaplace(F)
ans =
   cosh(7*t)-sinh(3*t)
```

5. 4*exp(-1/2*t)*sinh(1/2*t)

$\frac{dy}{dt} + y = 2U(t)$ 的 Laplace 变换是 sY+Y=2*(1/s)，化简得 Y=2/(s*(s+1))。

```
>> syms s
>> Y=2/(s*(s+1));
>> ilaplace(Y)
ans =
   4*exp(-1/2*t)*sinh(1/2*t)
```

6. -2*pi*dirac(2, w)

```
>> syms x;
>> fourier(x^2)
ans =
   -2*pi*dirac(2,w)
```

7. i*pi*(dirac(1, w - 1) + dirac(1, w + 1))

```
>> syms x;
>> fourier(x*cos(x))
ans =
   i*pi*(dirac(1,w-1)+dirac(1,w+1))
```

8. exp(-x)*heaviside(x)

```
>> syms w;
>> ifourier(1/(1+i*w))
ans =
   exp(-x)*heaviside(x)
```

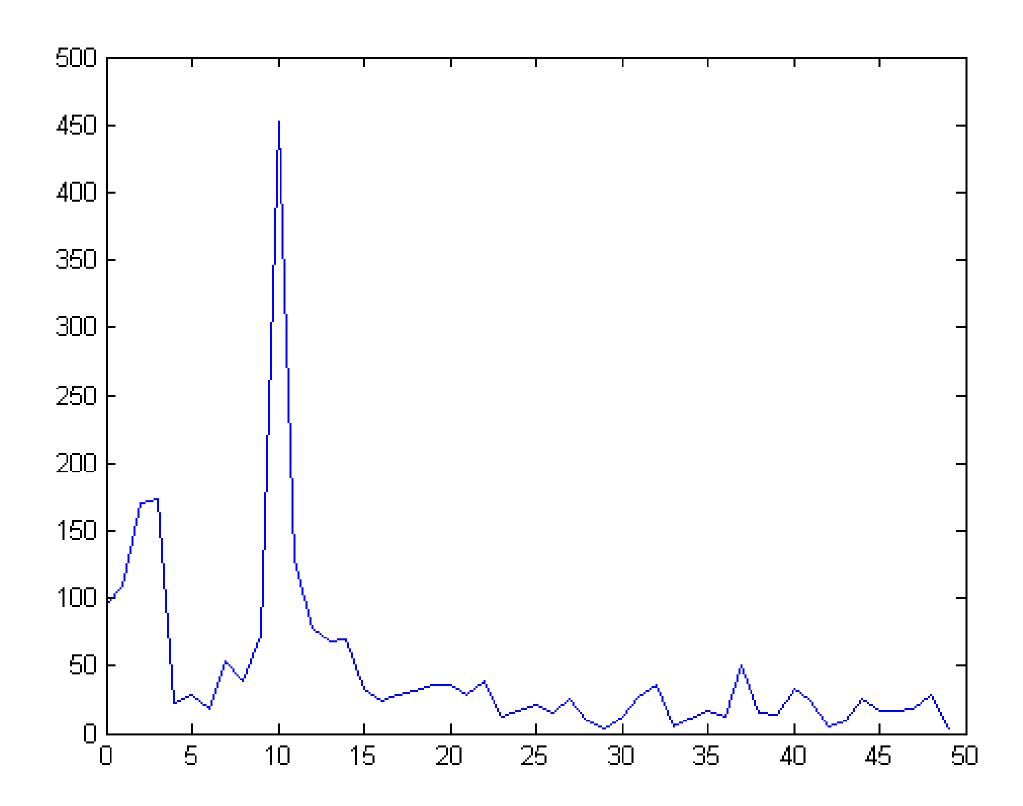
9. [7.0000, 3.0000 - 2.0000i, -5.0000, 3.0000 + 2.0000i]

```
>> a=[2,4,-1,2];
>> fft(a,4)
ans =
  7.0000  3.0000-2.0000i  -5.0000    3.0000+2.0000i
```

10. 447.7874 大概在 40 Hz.

译者注：我用下面的命令绘制了图象，可惜与所给的答案不一致，请读者指出下面的命令的错误之处。

```
>> t=0:0.01:5;
>> x=sin(pi*t)+2*sin(4*pi*t);
>> x_noisy=x+randn(size(t));
>> FT=fft(x_noisy,512);
>> f=t/0.01;
>> P=abs(FT);
>> plot(f(1:50),P(1:50));
```



第十章：曲线拟合

1. m = 11.8088, b = 191.5350.
```
>> age=[15,17,18,19,24,30,35,37];
>> maxweight=[330,370,405,420,550,580,600,580];
>> p=polyfit(age,maxweight,1);
>> m=p(1),b=p(2)
m =
   11.8088
b =
  191.5350
```
2. age_range = [15:1:37];
3. >> y = m*age_range + b;
4. 392.2850 磅
```
>> age_range=[15:1:37];
>> y=m*age_range+b;
>> y(find(age_range==17))
ans =
  392.2850
```
5. 479.3750 磅
```
>> N=length(maxweight)
N =
     8
>> MEAN=sum(maxweight)/N
MEAN =
  479.3750
```
6. 498.5643 磅（译者注：原版答案为 479.3750，有待进一步确认）
```
>> MEAN2=sum(y)/length(y)
```

```
MEAN2 =
  498.5643
```

7. S = 8.3122e + 004, A = 1.1184e + 004.

```
>> S=sum((maxweight-MEAN).^2)
S =
  8.3122e+004
>> w=m*age+b;
>> A=sum((w-maxweight).^2)
A =
  1.1184e+004
```

8. r2 = 0.8655.

```
>> r2=1-A/S
r2 =
    0.8655
```

9. $y = -0.8870x^2 + 58.0577x - 351.6032$

```
>> p=polyfit(age,maxweight,2)
p =
   -0.8870   58.0577 -351.6032
>> a = p(1);
>> b = p(2);
>> c = p(3);
```

10. 0.9896

```
>> MEAN=mean(maxweight)
MEAN =
  479.3750
>> S=sum((maxweight-MEAN).^2)
S =
  8.3122e+004
>> w=a*age.^2+b*age+c;
>> A=sum((w-maxweight).^2)
A =
  865.2110
>> r2=1-A/S
r2 =
    0.9896
```


第十一章：使用特殊函数工作

1. 3.5569e + 014

```
>> gamma(18)
ans =
  3.5569e+014
```

以下习题详细解答过程期待您的贡献……

2. 0.494
3.

```
>> x = (1:0.1:2)'; y = gamma(x); A = [x y]
A =
    1.0000    1.0000
    1.1000    0.9514
    1.2000    0.9182
    1.3000    0.8975
    1.4000    0.8873
    1.5000    0.8862
    1.6000    0.8935
    1.7000    0.9086
```

```
    1.8000    0.9314
    1.9000    0.9618
    2.0000    1.0000
```

5. `vol = @(n, r)r^n*pi^(n/2)/gamma(1 + n/2), 1.2791e + 003`
6. `1.9175e-009`
7. 图象如图 B11-7 所示。

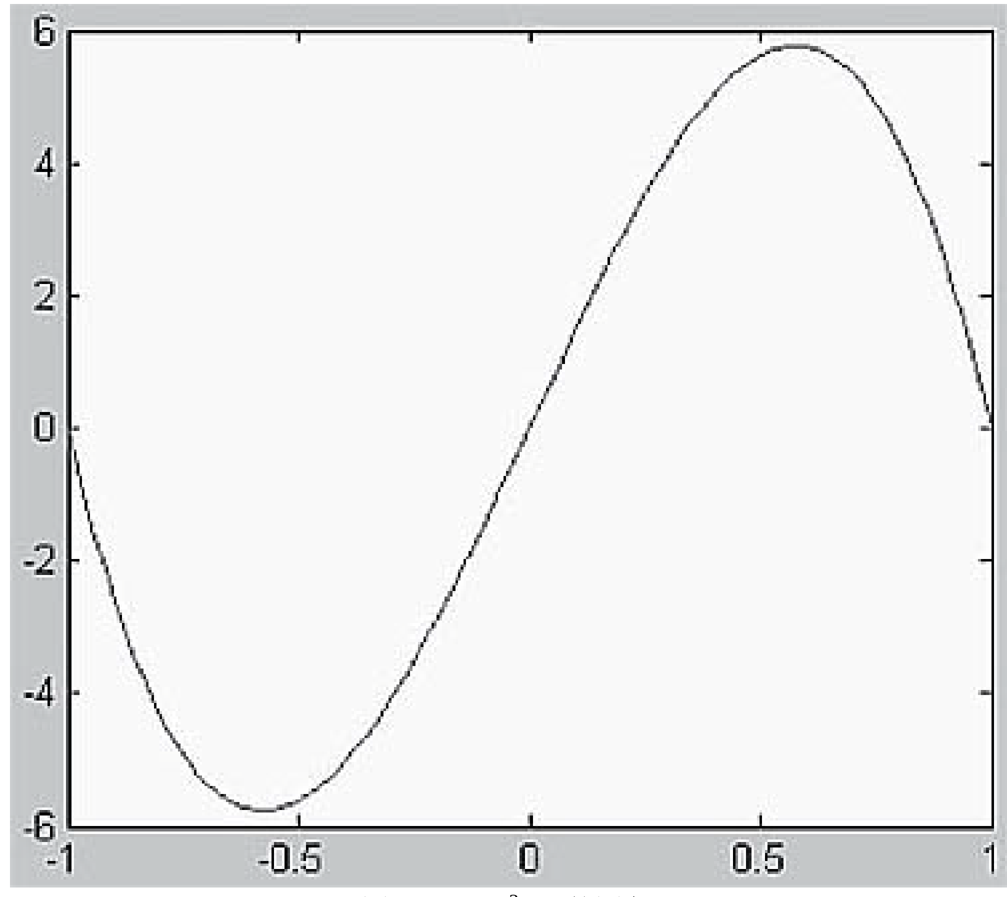


图 B11-7 $P_2^3(x)$ 的图象

8. `0.2995`
9. 表格是

```
     A =
            0  0.3550
       0.5000  0.2317
       1.0000  0.1353
       1.5000  0.0717
       2.0000  0.0349
       2.5000  0.0157
       3.0000  0.0066
       3.5000  0.0026
       4.0000  0.0010
       4.5000  0.0003
       5.0000  0.0001
```

10. $\dfrac{1}{3^{2/3}\Gamma(2/3)}=0.3550$

最终测试

1. 187.5
2. 2150
3. 16.35
4. 2*x*cos(pi*x) - x^2*sin(pi*x)*pi
5. 2/(x^2 - 9) - 4*x^2/(x^2 - 9)^2
6. 2*cos(x) - x*sin(x)
7. 使用展开命令得到 1 + 4*x + 6*x^2 + 4*x^3 + x^4.
8. x = 0, x = 1
9. 1
10. 极限不存在。.
11. 无穷大
12. 0
13. 0.0107
14. -7.99, -21.11
15. 470547/4*(x^(1/2) + 2)^96/x^(3/2) - 14553/4*(x^(1/2) + 2)^97/x^2 + 297/8*(x^(1/2) + 2)^98/x^(5/2)
16. solve
17. c
18. b
19. $\frac{5 \pm i\sqrt{11}}{2}$
20. (x - 4)*(x^2 + 4*x + 16)
21. c
22. 1
23. 9
24. 4.333
25. pretty(f)
26. d
27. a
28. b
29. c
30. b
31. d
32. c
33. c
34. a
35. d
36. b
37. a
38. a
39. d
40. d
41. b
42. a
43. c
44. b
45. b
46. a
47. b

48. a
49. a
50. c
51. b
52. 第十阶导数是：
```
-9101406417999/10000000000*exp(-1/10*t)*cos(2*t) +
24836027201/50000000*exp(-1/10*t)*sin(2*t)
```
53. b
54. b
55. c
56. a
57. b
58. b
59. a
60. b
61. `-atanh(1/(1 + x ^ 2) ^ (1/2))`
62. `-cos(x ^ 2)`
63. `1/5 * exp(x ^ 5)`
64. `sin(log(x))`
65. `log(x^2)*x - 2*x`
66. 4
67. 1/4
68. -inf
69. `cosh(b)-1`
70. 64/63
71. `-2/3 + exp(-1)`
72. 17/3
73. `5/16*exp(-4) + 3/16*exp(4)`
74. 0
75. d
76. c
77. `C1*exp(-2*t)`
78. `-1/2 + C1*exp(-2*t)`
79. `sin(2*t)*C2 + cos(2*t)*C1 + 1/2`
80. `C1*exp(-t) + C2*exp(1/2*t)*sin(1/2*3^(1/2)*t) + C3*exp(1/2*t)*cos(1/2*3^(1/2)*t)`
81. `sin(2^(1/2)*t)*C2 + cos(2^(1/2)*t)*C1 - 4/7*cos(3*t)`
82. `exp(t)`
83. `-2 + exp(t)`
84. `1/2*cos(t) + 1/2*sin(t) + 1/2*exp(-t)`
85. `exp(3^(1/2)*t)*(-1/6 + 1/6*3^(1/2)) + exp(-3^(1/2)*t)*(-1/6 - 1/6*3^(1/2)) + 1/3`
86. `-sin(t) - sin(1)*cos(1)/(sin(1)^2 - 1)*cos(t)`
87. `sin(t) + 5*cos(t) - 4`
88. `sin(t) + (5 - A)*cos(t) - 4 + A`
89. `4/5*sin(t) + 23/5*cos(t) - 4*exp(-2*t)*exp(2*t) + 2/5*exp(-2*t)`
90. `4/5*sin(t) + 23/5*cos(t) - 4*exp(-2*t)*exp(2*t) + 2/5*exp(-2*t)`
91. 解是：
```
x = exp(t)*(C1*cos(2*t) - C2*sin(2*t))
y = exp(t)*(C1*sin(2*t) + C2*cos(2*t))
```
92. `x = -exp(t)*sin(2*t), y = exp(t)*cos(2*t)`
93. 所求的逆是：

$$A^{-1} = \begin{pmatrix} -1/3 & 2/3 \\ 2/3 & -1/3 \end{pmatrix}$$

94. x = 3, y = 1/2
95. 3
96. 所求的逆是：

```
 0.2600   0.2400  0.0200
-0.2900   0.0400  0.1700
 0.0400  -0.0400  0.0800
```

97. 所求的 LU 分解是：

```
L =
0.3333   1.0000        0
1.0000        0        0
0.3333  -0.5000   1.0000
U =
3.0000   2.0000  -5.0000
     0  -2.6667   5.6667
     0        0  12.5000
```

98. 所求的本征值是：

```
1.6177 + 3.2034i
1.6177 - 3.2034i
7.7647
```

99. 31
100. 0.9045

芳名榜



网名	邮箱	贡献
bibofun	bibofun@163.com	本书的翻译、修订及维护
期待你的加入		